Grafting Key Trees: Efficient Key Management for Overlapping Groups

Joël Alwen[1], Benedikt Auerbach[*2], Mirza Ahad Baig[*2], Miguel Cueto[2], Karen Klein[†3], Guillermo Pascual-Perez[‡2], Krzysztof Pietrzak[*2], and Michael Walter[§4]

[1]AWS Wickr
alwenjo@amazon.com
[2]IST Austria, Klosterneuburg, Austria
{bauerbac, mbaig, miguel.cuetonoval, gpascual, pietrzak}@ist.ac.at
[3]ETH Zurich, Switzerland
karen.h.klein@protonmail.com
[4]Zama, Paris
michael.walter@zama.ai

September 10, 2021

Abstract

Key trees are often the best solution in terms of transmission cost and storage requirements for managing keys in a setting where a group needs to share a secret key, while being able to efficiently rotate the key material of users (in order to recover from a potential compromise, or to add or remove users). Applications include multicast encryption protocols like LKH (Logical Key Hierarchies) or group messaging like the current IETF proposal TreeKEM.

A key tree is a (typically balanced) binary tree, where each node is identified with a key: leaf nodes hold users' secret keys while the root is the shared group key. For a group of size N, each user just holds $\log(N)$ keys (the keys on the path from its leaf to the root) and its entire key material can be rotated by broadcasting $2\log(N)$ ciphertexts (encrypting each fresh key on the path under the keys of its parents).

In this work we consider the natural setting where we have many groups with partially overlapping sets of users, and ask if we can find solutions where the cost of rotating a key is better than in the trivial one where we have a separate key tree for each group.

We show that in an asymptotic setting (where the number m of groups is fixed while the number N of users grows) there exist more general key graphs whose cost converges to the cost of a single group, thus saving a factor linear in the number of groups over the trivial solution.

As our asymptotic "solution" converges very slowly and performs poorly on concrete examples, we propose an algorithm that uses a natural heuristic to compute a key graph for any given group structure. Our algorithm combines two greedy algorithms, and is thus very efficient: it first converts the group structure into a "lattice graph", which is then turned into a key graph by repeatedly applying the algorithm for constructing a Huffman code.

[*]Funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (682815 - TOCNeT).

[†]Supported in part by European Research Council (ERC) CoG grant 724307. A part of this work was conducted at IST Austria, funded by the ERC under the European Union's Horizon 2020 research and innovation programme (682815 - TOCNeT).

[‡]Funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No.665385.

[§]A part of this work was conducted at IST Austria, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (682815 - TOCNeT).

To better understand how far our proposal is from an optimal solution, we prove lower bounds on the update cost of continuous group-key agreement and multicast encryption in a symbolic model admitting (asymmetric) encryption, pseudorandom generators, and secret sharing as building blocks.

Contents

1 Introduction

Key trees. In various group communication settings, including multicast encryption [15, 16, 7] or group messaging protocols [4, 8], the most efficient constructions use a binary tree structure to manage keys. The general idea is to consider a balanced binary tree with edges directed from leaves to the root. One then identifies each node v with a key k_v (of a symmetric encryption scheme for multicast encryption and a public-key encryption scheme for group messaging). Each edge (u, v) corresponds to a ciphertext $\mathsf{Enc}_{k_u}(k_v)$ and each leaf node v with a user u_v. A user u_v will know the (secret) key k_v, and from the ciphertexts can then retrieve all the keys on the path from its leaf to the root ε. The root key k_ε is thus known to all users, and can be used for secure communication to or among the group members.

What makes this tree structure so appealing is the fact that in a group of size N, the key material of a user u can be completely rotated by replacing only the keys on the path from u to ε, which in a balanced tree has length at most $d = \lceil \log(N) \rceil$. Moreover, as the nodes in a tree all have indegree two, one only needs to compute two fresh ciphertexts for each new key (in practice just one as the new keys can be derived via a hash-chain).

These aspects are important as the number of keys a user requires basically defines the communication and computational efficiency of a key rotation, which is the main operation performed to add or remove users, or for a user to update their keys in order to recover from a potential compromise.

Groups. In this work we consider an extension of this setting to multiple groups. We are given a base set $[N] = \{1, \ldots, N\}$ of users with a set system $\mathcal{S} = \{S_1, \ldots, S_k\}$ (each $S_i \subseteq [N]$), and we ask for a key managing structure such that for any set $S_i \in \mathcal{S}$, the users in S_i share a group key. This is a natural and well motivated setting; consider for example a university, where one might want to have a shared key for all students attending particular lectures.

A trivial solution to this problem is to simply use a different key-tree for every group S_i, in this work we explore more efficient solutions.

Key-graphs beyond trees. For a set system $\mathcal{S}$ as above, instead of using disjoint trees, any directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the following properties is sufficient to maintain group keys:

1. Every user $i \in [N]$ corresponds to a source v_i (a node of indegree 0).
2. Every group $S_i \in \mathcal{S}$ corresponds to a sink v_{S_i} (a node of outdegree 0).
3. For every $S_i \in \mathcal{S}$ and $j \in [N]$, there is a directed path from v_j to v_{S_i} if and only if $j \in S_i$.
4. The indegree of any node is at most 2.

The first three properties ensure that any user $j \in [N]$ can learn the keys associated with the nodes of groups they are in. The last property is not really necessary, but it is without loss of generality in the sense that any graph can be turned into a graph with at most as large update cost (as we show in Section 3) where every node other than the leaves has indegree at most 2. We call this a key-derivation graph for $\mathcal{S}$.

Update cost. If we rotate the keys of a user i we need to replace all keys that can be reached from v_i, which we denote by $\mathcal{D}(v_i)$, and encrypt each new key under the keys of its co-path. We thus define the update cost of a user $i \in [N]$ as $\sum_{v \in \mathcal{D}(v_i)} \mathrm{indeg}(v) - 1$, which with item 4 above roughly simplifies to the number of v_i's descendants $|\mathcal{D}(v_i)|$. The update cost $\mathrm{Upd}(\mathcal{G})$ of a DAG $\mathcal{G}$ is the sum over the update cost of all its leaves, which is proportional to the average update cost of users.

Towards constructing more efficient key-derivation schemes when we have multiple overlapping groups, we thus address the problem of determining how small the update cost of a key-derivation for a given set system $\mathcal{S} = \{S_1, \ldots, S_k\}$ over $[N]$ can be, and how to find graphs which achieve, or at least come close to, this minimum.

Our contributions. We look at this problem from two perspectives. To get an insight on how much can be saved compared to the trivial solution, we first adapt a qualitative, asymptotic perspective, where we assume a fixed set system, but the number of users N goes to infinity while the relative size of the sets and intersections remains the same. We prove a lower bound on the update cost in this setting and give an algorithm computing graphs matching this bound.

As this solution turns out to be far from optimal for certain concrete set systems, we then also look at a quantitative non-asymptotic setting, where we consider concrete bounds and care about things like additive constants. We propose an algorithm that seems better equipped to handle such systems and prove upper and lower bounds on the update costs of graphs generated by it. Finally, we prove lower bounds on the update cost of any continuous group-key agreement scheme and multicast encryption scheme in a symbolic model.

1.1 The asymptotic setting

Given a set system $\mathcal{S} = (S_1, \ldots, S_k)$ over some base set $[n]$, we let $\mathcal{S}(N)$ denote the system with base set $[N]$ we get by considering each element in S with multiplicity N/n. E.g. if $\mathcal{S} = (\{1,2\},\{2,3\})$ then $\mathcal{S}(6) = (\{1,2,4,5\},\{2,3,5,6\})$.[1] Thus, as the number of users N grows the relative sizes of the groups and their intersections remain fixed.

Let $s_i := |S_i|/n$ denote the relative size of S_i and $s = \sum_{i=1}^{m} s_i$ be the average number of groups users are in. We assume wlog. that each user is in at least one group, implying $s \geq 1$. Let $\mathrm{Opt}(\mathcal{S})$ denote the update cost of the best key-graph for a set system $\mathcal{S}$ and $\mathrm{Triv}(\mathcal{S})$ the update cost of the Trivial algorithm (which makes a key-tree for every $S_i \in \mathcal{S}$). We will show that (the hidden constants in the big-Oh notation all depend on k, the number of groups).

$$\mathrm{Opt}(\mathcal{S}(N)) = N\log(N) + \Theta(N) \tag{1}$$

$$\mathrm{Triv}(\mathcal{S}(N)) = s \cdot N\log(N) - \Theta(N) \tag{2}$$

$$\text{thus } \frac{\mathrm{Triv}(\mathcal{S}(N))}{\mathrm{Opt}(\mathcal{S}(N))} = s - o(1) \tag{3}$$

As s is the average number of groups users are in, this shows that

> *asymptotically* (for a fixed set system $\mathcal{S}$ but with increasing number N of users) the update cost of an optimal key-derivation graph depends only on N (but not on $\mathcal{S}$). In this regime, the gain we get by using more cleverly chosen key-derivation graphs (as opposed to using a key-tree for every group) can be up to linear in s, the number of groups an average user is in, but not, say, the number of groups $|\mathcal{S}|$.

While we do not know how to efficiently find the best key graph for a given set system $\mathcal{S}$, in Section 4 we define a family $\mathcal{G}_{\mathrm{ao}}(\mathcal{S}(N))$ which is asymptotically optimal, i.e., matches Equation 1. Intuitively, it first partitions the universe of users $[N]$ into the sets of users that are members of exactly the same groups. More precisely, for $I \subseteq [k]$ let P_I be the set of users that are members of the groups specified by I. Then, the asymptotically optimal algorithm builds a balanced binary tree for every P_I, and in a second step connects the roots of these trees to the appropriate group keys by another layer of binary trees. For an illustration of the trivial and asymptotically optimal algorithms see Figure 1.

1.2 The non-asymptotic setting

Asymptotics can kick in slowly. The asymptotic setting gives a good idea about the efficiency we can expect once the number of users N is large compared to the number $k = |\mathcal{S}|$ of groups. Though it should be noted that this asymptotic effect can kick in only slowly: assume the artificial example where for some small base set $[n]$ we have a set system $\mathcal{S} = \{S_1, \ldots, S_k\}$ with $k = 2^n - 1$ groups where for every non-empty

[1] $\mathcal{S}(N)$ is only well defined if N/n is an integer, we ignore this technicality as we'll be interested in the case $N \to \infty$.

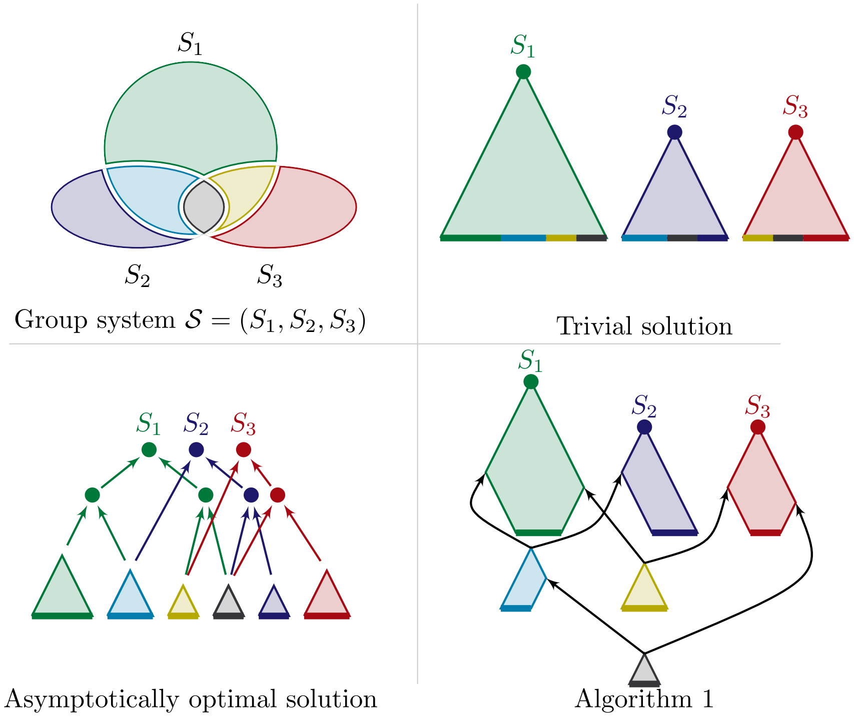

Figure 1: Key graphs for group systems. Top left; Venn diagram of the considered group system. Top right; trivial key graph using one balanced binary tree per group. Bottom left; Asymptotically optimal key graph using one balanced binary tree per partition P_I. Bottom right; asymptotically optimal key graph obtained using Algorithm 1. In the depictions of key trees the horizontal thick lines indicates the users' personal keys.

subset of users we have a group. Then each user is in 2^{n-1} groups and thus needs at least that many keys, and so the $\Theta(1)$ term in the asymptotic update cost $\log(N) + \Theta(1)$ of a single user is also at least 2^{n-1}. For the $\log(N)$ term to dominate we need $\log(N) \gg 2^{n-1}$, or $N \gg 2^{2^{n-1}}$, so the number of users needs to grow doubly exponential in the base set $[n]$.

Moving on to the non-asymptotic setting, consider a group system $\mathcal{S}$ for a fixed set of users $[N]$. The discussion above indicates that for $\mathcal{S}$ the asymptotic update cost per user of $\log(N)$ could be very far off the truth unless N becomes fairly large compared to the number of groups. This leaves the possibility that for concrete group systems where N is not huge relative to $\mathcal{S}$ already the trivial key-graph performs fairly well in practice. This, however, turns out to not be the case.

First, let us observe that the gap in update cost can never be larger than $\log(N)$, for any $\mathcal{S}$ over $[N]$

$$\mathrm{Triv}(\mathcal{S}) \leq \log(N) \cdot \mathrm{Opt}(\mathcal{S}) \tag{4}$$

To see this we observe that the update cost for every user $i \in [N]$ is at most a factor $\log(N)$ larger in the trivial solution: a user i that is in $s_i = |\{S \in \mathcal{S} \ : \ i \in S\}|$ groups has an update cost of at least s_i in any key graph, in particular in $\mathrm{Opt}(\mathcal{S})$, and at most $\sum_{S \in \mathcal{S}, i \in S} \log(|S|) \leq s_i \cdot \log(N)$ in the trivial key graph.

In Section 4.2 we will show that this is not merely a theoretical gap by giving an example of a natural system $\mathcal{S}$ for which the update costs of both the trivial and the asymptotically optimal algorithms match the gap of $\log(N)$.

A greedy algorithm based on Huffman codes. The discussion above indicates that for set systems mapping groups that we might encounter in practice, one shouldn't simply use an asymptotically optimal solution, but aim for a solution that is optimal, or at least close to optimal, for all instances.

Algorithm 1 that we propose in Section 5 is an algorithm for computing a key-graph given a set system $\mathcal{S}$. In a first step, the algorithm computes a "Boolean-lattice graph" for $\mathcal{S}$, and in a second iteratively runs the algorithm to compute Huffman Codes to compute the key graph. As the algorithm is basically a composition of greedy algorithms, it is very efficient. We leave it as an open question whether it really is optimal, and if not, whether there's an efficient (polynomial time) algorithm to compute $\text{Opt}(\mathcal{S})$ and find the corresponding key graph for a given $\mathcal{S}$ in general.[2]

We present Algorithm 1 in Section 5 and discuss its connection to Boolean lattices. Then, we derive concrete lower and upper bounds on its update cost, that can serve as a good estimate on how much it saves compared to the trivial algorithm and the asymptotically optimal algorithm of Section 1.1. We further show that Algorithm 1 and a class of algorithms generalizing the approach taken are optimal in the asymptotic setting. While the same is true for the algorithm discussed in Section 1.1, Algorithm 1 seems better suited for practical applications as key-derivation graphs constructed by it reflect the hierarchical structure inherent to such systems. An example of a key graph generated by it is in Figure 1.

Our analysis concerns static group systems, but in Section 6 we show how known techniques that allow adding and removing users from groups in the settings of continuous group-key agreement and multicast encryption for a single group can be adapted to key-derivation graphs generated by the greedy algorithm.

Lower bounds. To get a feeling how close to optimal our approach is, we prove a lower bound on the average update cost for arbitrary schemes for continuous group-key agreement (in Section 7) and multicast encryption (in Appendix B) that are based only on simple primitives such as encryption, pseudorandom generators, and secret sharing in a *symbolic security model*. This closely follows ideas from Micciancio and Panjwani [14] who considered such a symbolic model to analyze the worst-case update cost of multicast encryption schemes. We improve on their results by considering the setting of *multiple* potentially overlapping groups and proving a lower bound on the *average* communication complexity.

Our bound essentially shows that on average the cost of a user in any CGKA scheme or multicast encryption scheme for group system $S_1, \dots, S_k$ constructed from the considered primitives satisfies

$$\text{Upd}(\mathcal{G}) \geq \frac{1}{N} \cdot \sum_{\emptyset \neq I \subseteq [k]} |P_I| \cdot \log(|P_I|) \enspace ,$$

where $P_I \subseteq [N]$ is the set of users *exactly* in the groups specified by index set $I \subseteq [k]$. We consider it an interesting open question to either improve on this bound or to construct an algorithm matching it.

1.3 Related Work

In the setting of a single group key graphs have been used to construct secure multicast encryption, e.g. [15, 16, 7], and continuous group-key agreement (CGKA), e.g. [4, 8]. In the setting of multiple groups the approach to use binary trees for every set of users that are members of exactly the same groups similarly to the asymptotically optimal algorithm, has been suggested in [13, 17]. However, the trees are then combined in a way that induces an overhead that is linear in the number of trees.

In [9] Cremers et al. consider the post-compromise security guarantees of CGKA protocols for multiple groups. They show that in certain update scenarios protocols based on pairwise channels have better healing properties than protocols based on key trees, as updates in a single group also benefit all subgroups of it. We stress that these issues do not arise in our approach as updates in our setting are global and thus affect all groups the updating user is a member of.

The symbolic security model was first introduced by Dolev and Yao [10] and used by Micciancio and Panjwani [14] to prove worst case bounds on the update cost of multicast encryption schemes for a single group. In the context of CGKA schemes it was recently used by Bienstock et al. [6] who analyze the communication cost of concurrent updates in CGKA schemes for a single group.

[2]The question whether a polynomial time algorithm for computing $\text{Opt}(\mathcal{S})$ exists can be naturally asked in various ways. We discuss it in more detail in Section 8.

2 Preliminaries

2.1 Notation

Throughout the paper log denotes the logarithm with respect to base 2.

Graph notation. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed acyclic graph (DAG). To node $v \in \mathcal{V}$ we associate the sets $\mathcal{A}(v) = \{v' \in \mathcal{V} \mid \exists \text{ path from } v' \text{ to } v\}$ of *ancestors* of v, and $\mathcal{D}(v) = \{v' \in \mathcal{V} \mid \exists \text{ path from } v \text{ to } v'\}$ of *descendants* of v. Here, we allow paths of length 0 and hence $v \in \mathcal{A}(v)$ and $v \in \mathcal{D}(v)$. Let $\mathcal{G}' = (\mathcal{V}', \mathcal{G}')$ be a subgraph of $\mathcal{G}$ and $v \in \mathcal{V}'$. We denote the set of parents of v by $\mathcal{P}(v)$. The set of *co-parents* $\mathcal{CP}(v, \mathcal{G}') \subseteq \mathcal{V}$ of v with respect to $\mathcal{G}'$ in $\mathcal{G}$ is the set of vertices that are parents of v in $\mathcal{G}$ but not in $\mathcal{G}'$.

Probability distributions. Let X be a random variable that has outcomes $x_1, \ldots, x_\ell$ with probability $p_1, \ldots, p_\ell$. Then we denote by $\mathbb{E}[X]$ its expectation and by $H(X) = -\sum_{i=1}^{\ell} p_i \log(p_i)$ its Shannon entropy.

2.2 Huffman Codes

Given a collection $v_1, \cdots, v_\ell$ of disconnected leaves of weight $w_1, \ldots, w_\ell \in \mathbb{N}$ a *Huffman Tree* is constructed as follows. From the set $\{v_1, \ldots, v_\ell\}$ two nodes v_{i_1}, v_{i_2} with the smallest weights are picked. Then a node v and edges $(v_{i_1}, v), (v_{i_2}, v)$ are added to the graph. v's weight is set to $w_{i_1} + w_{i_2}$ and the set of nodes to be considered updated to $\{v_1, \ldots, v_\ell\} \cup \{v\} \setminus \{v_{i_1}, v_{i_2}\}$. This step is repeated until all leaves are collected under a single root.

Since all nodes have indegree 2 the Huffman tree defines a prefix-free binary code for $(v_1, \ldots, v_\ell)$. We will make use of the following property of Huffman Codes.

Lemma 1 (Optimality of Huffman Codes [11])**.** *Consider a Huffman tree $\mathcal{T}$ over leaves $v_1, \ldots, v_\ell$ of weight $w_1, \ldots, w_\ell \in \mathbb{N}$. Let $w = \sum_{i=1}^{\ell} w_i$ and let $U_\mathcal{T}$ denote the probability distribution that picks leaf v_i with probability w_i/w proportional to its weight. Then, if $\text{len}(U_\mathcal{T})$ denotes the random variable measuring the length of the path from a leaf picked according to $U_\mathcal{T}$ to the root, we have that the average length of such paths is bounded by*

$$H(U_\mathcal{T}) \leq \mathbb{E}[\text{len}(U_\mathcal{T})] \leq H(U_\mathcal{T}) + 1 \ .$$

3 Key-derivation Graphs for Multiple Groups

In this section we discuss key-derivation graphs for systems consisting of multiple groups. In Section 3.1 we briefly recall two applications of such graphs; continuous group-key agreement and multicast encryption. In Section 3.2 we formally define key-derivation graphs, discuss how key material in a graph is refreshed, and define its update cost.

3.1 Continuous Group-key Agreement and Multicast Encryption

Continuous group-key agreement. *Continuous group-key agreement* (CGKA) schemes [2] are an important building block in the construction of secure asynchronous group messaging schemes. As the name indicates, the goal of a CGKA scheme is to establish a common key that is to be used to secure the communication between members of a group. As groups can typically be long-lived, users need to also be able to frequently update the key material known to them, to on one hand, recover from a potential compromise and, on the other hand, ensure forward-secrecy of messages sent in the past.

In this work we are interested in the more general setting in which users $n \in [N]$ want to agree on keys for a system of groups $S_1, \ldots, S_k \subseteq 2^{[N]}$. After the groups have been established in a setup phase user n can use the procedure $\text{Upd}(n)$ to produce an update message that rotates the key material known to them, thus eliminating any keys that may have leaked during a compromise. This update message is broadcasted to the other users using the untrusted delivery server. Given their own secret keys, users are then able to retrieve the refreshed keys that should be known to them. A natural goal to aim for is to minimize the communication cost incurred by such update messages.

Naturally, one would like to additionally support dynamic operations, i.e., allow users to add and remove other users from groups in the system. While in this work we focus on the update costs of schemes for a system of static groups, in Section 6 we show that the known techniques of blanking and unmerged leaves used in the MLS protocol [4] can be adapted to schemes obtained from our approach.

Efficient CGKA protocols [4, 8] (in the single group setting) establish a key-derivation graph in the setup phase that, in turn, allows user to update at a cost that is logarithmic in the number of group members.[3] In Section 3.2 we formally define key-derivation graphs and discuss how the updating process works.

Multicast encryption. The goal of a *multicast encryption* scheme [15, 16, 7] is to establish a key for a group of users to enable them to decrypt ciphertexts broadcast to the group. Every user holds a personal long-term key, but opposed to CGKA there also exists a central authority that has access to all secret key material. After a setup phase, the central authority is able to add and remove users from the group by refreshing key material and broadcasting messages to the group. The central goal in the construction of multicast schemes is to minimize the communication complexity incurred by such operations. Typically, multicast encryption schemes also rely on key-derivation graphs.

As in the case of CGKA, we are interested in the more general setting of a system of potentially overlapping groups of users.

3.2 Key-derivation Graphs

We now discuss key-derivation graphs. In our exposition we will focus on graphs for continuous group-key agreement. At the end of the section we discuss the differences to graphs for multicast encryption.

Consider a set of parties $[N]$ and a collection $\mathcal{S} \subseteq 2^{[N]}$ of subgroups of $[N]$. A key-derivation graph (kdg) for $[N]$ and $\mathcal{S}$ organizes key pairs in a way that allows the members of a particular subgroup to agree on a key, and further enables parties to refresh the key material known to them. Every node v in the graph is associated to a key pair $(\mathsf{pk}_v, \mathsf{sk}_v)$ of a public-key encryption scheme $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, and edges (v, v') indicate that parties with access to sk_v also posses $\mathsf{sk}_{v'}$. The personal keys of users correspond to sources and every group is represented by a node that holds the corresponding secret group key. We formalize the structural requirements on the graph in the multi-group setting as follows.

Definition 1. *Let $N \in \mathbb{N}$, $\mathcal{S} \subseteq 2^{[N]}$, and $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ a DAG. We say that $\mathcal{G}$ is a* key-derivation graph *for universe of elements $[N]$ and groups $\mathcal{S}$ if*

1. *For every $n \in [N]$ there exists a source $v_n \in \mathcal{V}$ and for every $S \in \mathcal{S}$ there exists a node $v_S \in \mathcal{V}$. We further require that $v_n \neq v'_n$ for $n \neq n'$.*
2. *For $n \in [N]$ and $S \in \mathcal{S}$ we have $v_S \in \mathcal{D}(v_n)$ exactly if $n \in S$.*

In the definition above node v_n correspond to user n's personal key, and nodes v_S to group keys. The second property encodes correctness and security, intuitively saying that n is able to derive the group key of S exactly if $n \in S$.

Updates. Let $\mathcal{G}$ be a key-derivation graph for $[N]$ and $\mathcal{S}$. If party n wants to perform an update she has to refresh all key-material corresponding the subgraph $\mathcal{D}(v_n)$ known to her and communicate the change to the other parties. To this end she picks a spanning tree $\mathcal{T}_n = (\mathcal{V}', \mathcal{E}')$ of $\mathcal{D}(v_n)$, as well as a random seed Δ_{v_n}. Then starting from the source v_n, if v' is the ith child of node v she defines the seed of v' as $\Delta_{v'} = \mathsf{H}(\Delta_v, i)$, where H is a hash function. $\Delta_{v'}$ is then used to derive a new key-pair $(\mathsf{pk}_{v'}, \mathsf{sk}_{v'}) \leftarrow \mathsf{KGen}(\Delta_{v'})$ for v'.

[3] In order to ensure authenticity of update messages and to prevent the server from sending users inconsistent update messages these protocols employ additional techniques. We leave the question how to adapt these to key-derivation graphs for multiple groups to future work (See Section 8).

Finally, n for every $v \in \mathcal{V}'$ and every co-parent $v' \in \mathcal{CP}(v, \mathcal{T}_n)$ computes the ciphertext $c_{v,v'} = \mathsf{Enc}(\mathsf{pk}_{v'}, \Delta_v)$. The set of all ciphertexts together with the set of new public keys forms the update message. Finally, n deletes all seeds Δ_v.

We now show that the construction preserves correctness, i.e., users $n' \neq n$ are able to deduce all new secret keys in $\mathcal{D}(v_{n'})$ from the update message and thus in particular the group keys of all groups they are a member of. To this end, let $v \in \mathcal{D}(v_n) \cap \mathcal{D}(v_{n'})$. Then there exists a path $(v_{n'} = v_1, \ldots, v_\ell = v)$ in $\mathcal{D}(v_{n'})$. Let i be maximal with $v_i \notin \mathcal{D}(v_n)$ (Note that such i must exist as $v_{n'}$ is a source). By maximality of i the node v_i must be a coparent of v_{i+1} with respect to $\mathcal{D}(v_n)$. Thus, the update message contains an encryption of Δ_{v_i+1} to pk_{v_i}. As sk_{v_i} was not replaced by the update and is known to n' the user can recover $\Delta_{v_{i+1}}$ and in turn $\mathsf{sk}_{v_{i+1}}$. Now, n' can recover the remaining $\Delta_{v_{i+2}}, \ldots, \Delta_{v_\ell}$ and the corresponding secret keys as the seeds were either derived by hashing or, in the case that v_{j+1} is a coparent of v_j with respect to $\mathcal{D}(v_n)$, encrypted to the new key pk_{v_j}, the secret key of which was already recovered by n'.

Update cost. Using the size of ciphertexts as a unit, the update cost of n is given by $\mathrm{Upd}(n) = \sum_{v \in \mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)| = \sum_{v \in \mathcal{T}_n} (|\mathcal{P}(v)| - 1)$. Note that this quantity is independent of the particular choice of spanning tree $\mathcal{T}_n$. In this work we are interested in minimizing the *average* update cost, assuming that every user updates with the same probability. We define the *total update cost* $\mathrm{Upd}(\mathcal{G}) = \sum_{n \in [N]} \mathrm{Upd}(n)$ of $\mathcal{G}$. Note that $\mathrm{Upd}(\mathcal{G})/N$ is the average update cost of a user, and we can thus focus on trying to minimize $\mathrm{Upd}(\mathcal{G})$, which will allow for easier exposition. The following lemma shows that we can restrict our view to graphs in which every non-source has indegree 2. Note, that for graphs $\mathcal{G}$ with this property we have $|\mathcal{CP}(v, \mathcal{T}_n)| = 1$ for every n, $\mathcal{T}_n$, and $v \in \mathcal{T}_n$ that is not a source and thus in this case we can compute the update cost as

$$\mathrm{Upd}(\mathcal{G}) = \sum_{n \in [N]} (|\mathcal{T}_n| - 1) = \sum_{n \in [N]} (|\mathcal{D}(n)| - 1) = \sum_{n \in [N]} |\mathcal{D}(n)| - N \ . \tag{5}$$

Lemma 2. *Let $n \in \mathbb{N}$, $\mathcal{S} \subseteq 2^{[N]}$, and $\mathcal{G}$ a key-derivation graph for $[N]$ and $\mathcal{S}$. Then there exists a key-derivation graph $\mathcal{G}'$ for $[N]$ and $\mathcal{S}$ satisfying $\mathrm{Upd}(\mathcal{G}') \leq \mathrm{Upd}(\mathcal{G})$ such that for every non-source $v \in \mathcal{V}'$ we have $\mathrm{indeg}(v) = 2$.*

Proof. We first show that we can iteratively decrease the indegree of nodes in $\mathcal{G}$ in a way that preserves correctness and only improves the update cost. Thus let $v \in \mathcal{V}$ and $\mathcal{P}(v) = \{v_1, \ldots, v_k\}$ with $k \geq 3$ and consider the graph $\mathcal{G}'$ with $\mathcal{V}' = \mathcal{V} \cup \{v'\}$ and $\mathcal{E}' = \mathcal{E} \cup \{(v_1, v',), (v_2, v'), (v', v)\} \setminus \{(v_1, v), (v_2, v)\}$. Note that the correctness requirements of Definition 1 are unaffected by this modification. Let $n \in [N]$. If $v \notin \mathcal{D}(n)$ n's update cost remains the same after the change. Thus, assume $v \in \mathcal{D}(v)$. In $\mathcal{G}$ we have $|\mathcal{P}(v)| = k$. In $\mathcal{G}'$, on the other hand, $|\mathcal{P}(v)| = k - 1$ and $|\mathcal{P}(v') = 2|$. Thus if n's spanning trees in $\mathcal{G}$ uses edge (v_1, v) or (v_2, v), then her update cost remains unchanged in $\mathcal{G}'$. If, on the other hand, neither (v_1, v) nor (v_2, v) lie in n's spanning trees, then her update cost decreases by one. after repeating the step sufficiently many times we end up with a graph $\mathcal{G}'$ with $\mathrm{indeg}(v) \leq 2$ for all $v \in \mathcal{V}'$.

Finally note that non-sources with indegree 1 can be simply merged with their parent, which does not affect correctness and update cost of the graph. ☐

Key-derivation graphs for multicast encryption. Opposed to kdgs for CGKA key-derivation graphs for multicast encryption rely on symmetric encryption. Let (E, D) be a symmetric encryption scheme. Every node v in a kdg $\mathcal{G}$ for $[N]$ and $\mathcal{S}$ is associated to a key k_v, and an edge (v, v') indicates that a party with access to k_v allows knows $\mathsf{k}_{v'}$. We require structural requirements on $\mathcal{G}$ that are analogous to Definition 1. Updates with respect to leaf v_n, which for multicast encryption are computed by the central authority, and their update cost are defined analogous to the setting of CGKA as well.

While the main goal of multicast encryption is not to recover from compromise of keys by updating, but instead to allow the central authority to dynamically change the structure of the groups $S_1, \ldots, S_k$ the notion of an update with respect to a leaf v_n still turns out to be useful. Assume that the central authority performed an update for v_n starting with seed Δ. We can distinguish two cases. If Δ is not known to the owner n of leaf v_n then n lost access to all keys corresponding to $\mathcal{D}(v_n)$. Thus, by updating the central authority can remove a party from *all* groups they are a member of. Assume on the other hand that the leaf was previously unpopulated and that Δ can be derived from n's long term key. Then n gained access to all group keys that can be reached from v_n. In Section 6 we discuss how updates can be used as the basic building block of implementing more fine grained operations, i.e., adding or removing a user from particular group S_i. The efficiency of these operations is significantly determined by the update cost as defined in this section.

3.3 Security

The main focus of this work is to investigate the communication complexity of key-derivation graphs for group systems. We do not give formal security proofs in this work. The structural requirements on kdgs and definition of update procedures are chosen with the goal of the resulting CGKA to achieve *post-compromise forward-secrecy* (PCFS) [3] roughly corresponding to *post-compromise security* (PCS) and *forward-secrecy* (FS) simultaneously. In the following paragraphs we provide an intuition on the security properties of kdgs. For ease of exposition we will discuss PCS and FS separately instead of PCFS.

Note that CGKA schemes constructed from kdgs employ further mechanisms to ensure authenticity and prevent a malicious sever to send users inconsistent update messages. We consider the construction of such mechanisms as well as a formal security analysis of kdgs to be important open questions for future work.

Preserving the graph invariant. We first discuss how updates preserve the invariant, that users n know exactly the secret keys corresponding to $\mathcal{D}(v_n)$, which by Condition 2 of Definition 1 implies that n will never be able to derive a group key for some group they are not a member of. Note that if n is the updating user then they will only replace keys in $\mathcal{D}(v_n)$. If n receives an update message, on the other hand, then they will only be able to recover a key sk_v if either the corresponding seed Δ_v was encrypted to a key known to n or if Δ_v was derived by hashing from a seed $\Delta_{v'}$ recoverable by n. By iteratively applying this argument to $\Delta_{v'}$ we obtain that there must exist some $\Delta_{v''}$ that was encrypted to a key known to n such that v'' has a path to v. Thus, it must hold that $v \in \mathcal{D}(v_n)$. (Note that the one-wayness of the used hash function ensures that seeds derived by hashing can only be recovered from each other in the correct direction.)

Post-compromise security. The goal of PCS is to allow users whose secret state has been exposed to recover from this exposure by performing an update. Using the example of a single compromised user we now discuss how kdgs for group systems achieve this property. Assume that an adversary knows exactly the secret state of user n, i.e., all keys sk_v for $v \in \mathcal{D}(v_n)$, and that n then performs an update. Then the adversary is not able to deduce any of the replaced keys: Note that the initial random seed Δ_{v_n} is not encrypted to any key and thus cannot be leaked to the adversary. Thus, all other seeds Δ_v can only be derived by the adversary if Δ_v itself, or a seed from which Δ_v was derived by iterated hashing was encrypted to a key known to the adversary. However, the adversary only knows the keys corresponding to $\mathcal{D}(v_n)$ before the update, and those keys were replaced by freshly sampled ones before computing the ciphertexts. Thus, seeds are either encrypted to "old" keys not known to the adversary or new keys, and in turn after the update all keys are secure again.

Forward secrecy. Forward secrecy requires that compromising a user's secret state does not allow the adversary to recover previous group keys. In key-derivation graphs old keys get deleted over time providing a limited form of forward-secrecy. Concretely, if a user n is corrupted all group keys before their last update remain secure. This holds, since seeds that were generated before this point in time and can be used to recover group keys were encrypted to keys no longer in n's memory. Note however, that group keys generated in between n's last update and the time of n's corruption might leak to the adversary. For example, a seed from which such keys can be derived might have been encrypted to the key sk_{v_n} which remained unchanged until the corruption.

Improved forward secrecy using supergroups. CGKA constructions relying on kdgs like TreeKEM [5] rely on an additional mechanism to improve their forward-secrecy guarantees. Instead of directly using group keys sk_{v_S} to communicate within the group these keys are used to derive a so called *application secret* K that serves as the symmetric key for group communication. Whenever an update occurs, the new application secret of S is computed as $\mathsf{H}_2(\mathsf{sk}_{v_S}, K)$ the output of a hash function on input of the new group key and the previous application secret. Then the old application secret is deleted from memory. The effect of this is, that when a user's state leaks (including the current application secret K_t) no old application secret K_i can be recomputed from old update messages, unless K_{i-1} was already known to the adversary by former corruptions. In short, users gain the advantage of forward secrecy not only by issuing but also by processing updates of other users in S.

In the setting of a group system $\mathcal{S}$ we can further improve on this: Consider some group $S \in \mathcal{S}$ and let $S_1, \ldots, S_\ell$ be the maximal (with respect to inclusion) groups in $\mathcal{S}$ that contain S. We denote the application secrets for S and the S_i by K_S and K_{S_i} respectively. Now, whenever a member of any of the S_i issues an update the application secret of S is updated to $K_S \leftarrow \mathsf{H}_2(\mathsf{sk}_{v_S}, K_{S_1}, \ldots, K_{S_\ell})$.[4] Note that for every i since $S \subseteq S_i$ all members of S do indeed have access to K_{S_i} and thus are able to compute K_S, and that an update by users in S implies that all S_i are updated as well. The effect of this modification is that even updates by users *outside* of S—more precisely in any of the sets $S_i \setminus S$—imply forward secrecy of users in S. Note that this is in particular helpful in the case where $|S| \ll |S_i|$ and updates in the large group occur much more frequently than in the small group, for example in the case of two members of a large group having a private conversation.

3.4 The Trivial Algorithm

To construct a key-derivation graph for a single group S the parties $n \in S$ are typically arranged as the leaves of a balanced binary tree $\mathcal{T}$. The tree's root serves as the group key. In this case the length of paths from leaf to root is at most $\lceil \log(|S|) \rceil$ and in turn $\mathrm{Upd}(\mathcal{T}) \leq |S| \cdot \lceil \log(|S|) \rceil$. On the other hand, $\mathcal{T}$ defines a prefix-free binary code for the set S. Thus, by Shannon's source coding theorem the average length of paths from leaf to root is at least $\log(|S|)$ which implies $\mathrm{Upd}(\mathcal{T}) \geq |S| \cdot \log(|S|)$.

An algorithm for multiple groups. A trivial approach to construct a key derivation graph for parties $[N]$ and group system $\mathcal{S} = \{S_1, \ldots, S_k\}$ is to simply apply the method described above to all S_i in parallel. That is, for $i \in [k]$ construct a balanced binary tree $\mathcal{T}_i$ with $|S_i|$ leaves such that for $n \in [N]$ the node v_n is a leaf of exactly the trees $\mathcal{T}_i$ with $n \in S_i$. Let $\mathcal{G}$ denote the resulting graph. The conditions of Definition 1 clearly hold and we can bound the total update cost of $\mathcal{G}$ by

$$\sum_{i \in [k]} |S_i| \cdot \log(|S_i|) \leq \mathrm{Upd}(\mathcal{G}) \leq \sum_{i \in [k]} |S_i| \cdot \lceil \log(|S_i|) \rceil \quad .$$

Further, the update cost of a single user $n \in [N]$ is bounded by $\mathrm{Upd}(n) \leq \sum_{i : n \in S_i} \lceil \log(|S_i|) \rceil$.

4 Key-derivation Graphs in the Asymptotic Setting

In this section we investigate the update cost of key-derivation graphs for multiple groups in an asymptotic setting. More precisely, for a system consisting of a fixed number of groups we consider the setting in which the number of users tends to infinity while the relative size of the groups stays constant. In Section 4.1 we first compute the asymptotically optimal update cost of key-derivation graphs and then show that the trivial algorithm does not achieve it. We then present an algorithm achieving the optimal update cost. In Section 4.2 we show that both approaches can perform badly for *concrete* group systems.

[4] Regarding PCFS it might even be advantageous to include $K_{S'}$ for *all* $S' \supseteq S$.

4.1 Key-derivation Graphs in the Asymptotic Setting

We investigate the update cost of key derivation graphs in an asymptotic setting. That is, we consider N parties that form a subgroup system $\mathcal{S} = \{S_1, \ldots, S_k\}$ and fix values $p_I \in [0, 1]$ for $I \subseteq [k]$ that indicate the fraction of users that are members of exactly the groups specified by I.

More precisely, let $k \in \mathbb{N}_{\geq 2}$ be fixed and let $\{p_I\}_{I\subseteq[k]}$ be such that $\sum_{I\subseteq[k]} p_I = 1$. For $N \in \mathbb{N}$ let $\mathcal{S}(N) = \{S_1(N), \ldots, S_k(N)\} \subseteq 2^{[N]}$ be a subgroup system that satisfies $|P_I(N)| = N \cdot p_I$ for all I, where $P_I(N) = \bigcap_{i\in I} S_i(N) \setminus \bigcup_{j\in[k]\setminus I} S_j(N)$ is the set of users exactly in the groups specified by I.[5] Throughout this section we assume that $p_\emptyset = 0$, i.e., every user is in at least one group, and that at least two groups are non-empty. We are interested in the update cost of key-derivation graphs for $\mathcal{S}(N)$ when N tends to infinity.

Lower bound in the asymptotic setting. We first compute a lower bound on the update cost of kdgs in the asymptotic setting. The bound follows from the following combinatorial result on *concrete* graphs that will also turn out to be useful for our symbolic lower lower bound of Section 7. Recall that for graphs $\mathcal{G}' \subseteq \mathcal{G}$ and a vertex v the set $\mathcal{CP}(v, \mathcal{G}')$ is the set of co-parents of v with respect to $\mathcal{G}'$ in $\mathcal{G}$.

Lemma 3. *Let $M \in \mathbb{N}$ be fixed, $S_1, \ldots, S_k \subseteq [M]$, and let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a DAG such that there exist pairwise disjoint sets of sources V_n, $n \in [M]$, and nodes v_{S_i}, $i \in \{1, \ldots, k\}$ such that*

$$n \in S_i \quad \Rightarrow \quad \exists v_n \in V_n \text{ such that there is a path from } v_n \text{ to } v_{S_i} \ .$$

Further let $\mathcal{T}_n$ be a spanning forest of $\mathcal{D}(V_n) = \bigcup_{v_n\in V_n} \mathcal{D}(v_n)$. Then

$$M \cdot \mathbb{E}\Big[\sum_{v\in\mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)| \Big] \geq \sum_{\emptyset\neq I\subseteq[k]} |P_I| \cdot \log(|P_I|) \ ,$$

where the expectation is to be understood with respect to the uniform distribution on $[N]$.

Proof. As a first step we show that we may assume that all V_n consist of a single source v_n. Indeed, we could replace $\mathcal{G}$ with a graph $\mathcal{G}'$ that for every n has an additional source v_n which has outgoing edges to all elements of V_n. As now all former sources have indegree 1 and all other nodes are unaffected $\sum_{v\in\mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)|$ is the same in both graphs, and every bound on for $\mathcal{G}'$ carries over to $\mathcal{G}$.

Further, using the same argument as in the proof of Lemma 2 we can replace $\mathcal{G}'$ by a graph $\mathcal{G}''$ satisfying the same correctness properties as $\mathcal{G}'$ such that all non-sources v of $\mathcal{G}''$ satisfy $\text{indeg}(v) = 2$ and $\sum_{v\in\mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)|$ can only decrease, implying that every bound for $\mathcal{G}''$ carries over to $\mathcal{G}'$ and in turn to $\mathcal{G}$. Thus assume that $\mathcal{G}$ satisfies $V_n = \{v_n\}$ for all n and that all non-sources have indegree 2.

Note that

$$\begin{aligned} M \cdot \mathbb{E}[\sum_{v\in\mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)|] &= \sum_{n\in[N]} \sum_{v\in\mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)| \\ &= \sum_{\emptyset\neq I\subseteq[k]} \sum_{n\in P_I} \sum_{v\in\mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)| \ , \end{aligned}$$

where we used that the P_I form a partition of $[M]$ and that $P_\emptyset = \emptyset$. Thus, to prove the lemma it suffices to show that for every nonempty $I \subseteq [k]$ we have $\sum_{n\in P_I} \sum_{v\in\mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)| \geq |P_I| \cdot \log(|P_I|)$. Fix a nonempty index set $I \subseteq [k]$ and let $i \in I$. By assumption for all $n \in P_I$ there exists a path $\mathcal{P}$ from v_n to v_{S_i}. Thus there exits a subgraph $\mathcal{G}'$ of $\mathcal{G}$ in which every v_n has exactly one path to v_{S_i}.

As all v_n are sources and $\text{indeg}(v) \leq 2$ for all $v \in \mathcal{G}'$ the graph $\mathcal{G}'$ defines a prefix-free binary code for the set P_I. By Shannon's source coding theorem this implies that the average length of the paths from source to sink in $\mathcal{G}'$ is at least $H(U_{P_I}) = \log(|P_I|)$, where U_{P_I} denotes the uniform distribution over P_I. Summing over all elements of P_I we obtain

$$\sum_{n\in P_I} \sum_{v\in\mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)| = \sum_{n\in P_I} (|\mathcal{T}_n| - 1) \geq |P_I| \cdot \log(|P|_I) \ ,$$

[5] $\mathcal{S}(N)$ is only well defined if $N \cdot p_I$ is an integer for all I, we ignore this technicality as we are interested in the case $N \to \infty$.

where in the equality we used that all non sources have indegree 2 and in the inequality that $\mathcal{T}_n$ contains paths of average length $\log(P_I)$.

□

Note that Lemma 3 in the case $|V_n| = 1$ for all n can be seen as a lower bound on the total update cost of key-derivation graphs as defined in Section 3 since $M \cdot \mathbb{E}[\sum_{v \in \mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)|] = \sum_{v \in \mathcal{T}_n} |\mathcal{CP}(v, \mathcal{T}_n)|$.

Turning to the asymptotic setting we have

$$\begin{aligned}\sum_{I \subseteq [k]} N \cdot p_I \cdot \log(N \cdot p_I) &= N \cdot \sum_{I \subseteq [k]} p_I \log(N) + N \cdot \sum_{I \subseteq [k]} p_I \log(p_I) \\ &= N \log(N) + N \cdot \sum_{I \subseteq [k]} \log(p_I) = N \log(N) + \Theta(N) \ ,\end{aligned}$$

where we used that $\sum_I p_I = 1$. As we will show below, there exist key-derivation graphs matching this bound. We conclude that the optimal update cost in the asymptotic setting only depends on the overall number of users but not the particular set system:

$$\mathrm{Opt}(\mathcal{S}(N)) = N \log(N) + \Theta(N) \ .$$

Note, however, that the term $\Theta(N)$ hides a constant (with respect to N), that can be *exponential in k*.

Asymptotic update cost of the trivial algorithm. The trivial algorithm constructs a separate balanced binary tree for every group $S_i(N)$. For $i \in [k]$ let s_i be such that $N \cdot s_i = |S_i(N)|$ and further let $s = \sum_{i=1}^{k} s_i$ be the average number of groups a user are member of. Then we can bound the update cost $\mathrm{Triv}(\mathcal{S}(N))$ of the trivial algorithm in the asymptotic setting as follows, showing that is does not match the optimal cost..

Claim 1. *For $I \subseteq [k]$ let $p_I \in [0, 1]$ be such that $\sum_{I \subseteq [k]} p_I = 1$ and $p_\emptyset = 0$. Let $\mathcal{S}(N)$ be the corresponding group system and s_i, s as defined above. Then*

$$\mathrm{Triv}(\mathcal{S}(N)) = s \cdot N \log(N) + \Theta(N) \ .$$

Proof. As discussed in Section 3.4, key-derivation graphs $\mathcal{G}$ for $\mathcal{S}(N)$ constructed by the trivial algorithm satisfy

$$\sum_{i \in [k]} |S_i(N)| \cdot \log(|S_i(N)|) \leq \mathrm{Upd}(\mathcal{G}) \leq \sum_{i \in [k]} |S_i(N)| \cdot \lceil \log(|S_i(N)|) \rceil \ .$$

Thus, on the one hand,

$$\begin{aligned}\mathrm{Triv}(\mathcal{S}(N)) &\geq \sum_{i=1}^{m} (s_i \cdot N) \cdot \log(N \cdot s_i) = \sum_{i=1}^{m} (s_i \cdot N) \cdot \log(N) + \log(s_i) \\ &= \sum_{i=1}^{m} s_i \cdot N \log N + \sum_{i=1}^{m} s_i \cdot N \cdot \log(s_i) = s \cdot N \log(N) + \Theta(N)\end{aligned}$$

and, on the other hand,

$$\begin{aligned}&\mathrm{Triv}(\mathcal{S}(N)) \\ &\leq \sum_{i=1}^{m} (s_i \cdot N) \cdot \log(N \cdot s_i) + O(N) = \sum_{i=1}^{m} (s_i \cdot N) \cdot \log(N) + \log(s_i) + O(N) \\ &= \sum_{i=1}^{m} s_i \cdot N \log N + \sum_{i=1}^{m} s_i \cdot N \cdot \log(s_i) + O(N) \leq s \cdot N \log(N) + O(N) \ .\end{aligned}$$

□

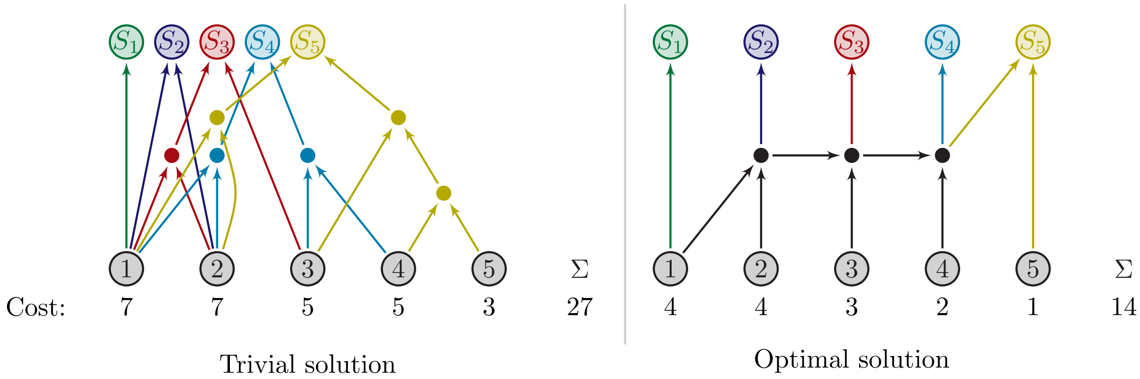

Figure 2: Illustration of $\mathrm{Triv}(\mathcal{S}_N^{\uparrow})$ (left) and $\mathrm{Opt}(\mathcal{S}_N^{\uparrow})$ for $N = 5$. For each user, the update cost (i.e., the indegree 2 nodes reachable) is indicated.

An asymptotically optimal graph. We will sketch how to construct an asymptotically optimal key graph $\mathcal{G}_{\mathrm{ao}}(N)$ for a given set system $\mathcal{S}$ over $[n]$. In a first step the algorithm for every I with $P_I(N) \neq \emptyset$ constructs a balanced binary tree with root v_I using as leafs the elements of $P_I(N)$. Then, in a second step for every group $S_i(N)$ it builds a balanced binary tree with root v_{S_i} using as leafs the nodes $\{v_I \mid I : i \in I\}$. An illustration of the algorithm's working principle is in Figure 1. Correctness of the construction follows by inspection.

To bound the update cost $\mathrm{Upd}(\mathcal{G}_{\mathrm{oa}}(N))$ we split it in two parts; the first accounts for the contribution of the nodes generated during the first step, the second for the contribution of the second step. As $\sum_I p_I = 1$, the first part contributes at most $\sum_{I \subseteq [k]} p_I \cdot N \cdot \log(N \cdot p_I) \leq N \cdot \log N$, while the contribution of the second part for every single user is constant as $\{v_I\}$ is independent of N, implying that with respect to the total update cost it is $\Theta(N)$. Thus, overall we get $\mathrm{Upd}(\mathcal{G}_{\mathrm{oa}}(N)) \leq N \cdot \log N + \Theta(N)$ matching the optimal update cost.

4.2 Update Cost for Concrete Group Systems

Now consider a concrete group system $\mathcal{S} = \{S_1, \ldots, S_k\}$ for a fixed set of users $[N]$. As already discussed in Section 1.2, it is possible that the number k of groups can be as large as $2^N - 1$. Thus, for concrete group systems the asymptotic update cost per user of $\log(N)$ (that contains hidden constants dependent on k) derived in Section 4.1 could be very far off the truth unless N becomes fairly large compared to the number of groups. This leaves the possibility that in the case where N is not huge relative to k already the trivial key-graph performs fairly well in practice. In this section we show that this is not the case by giving an example where not only the trivial key-graph (which has a balanced tree for every set), but also our asymptotically optimal $\mathcal{G}_{\mathrm{oa}}$ perform poorly.

Recall that by Equation 4 the update costs of the trivial and optimal solutions always satisfy $\mathrm{Triv}(\mathcal{S}) \leq \log(N) \cdot \mathrm{Opt}(\mathcal{S})$. The above argument seems very loose, but we show an example where we indeed have a gap of $\approx \log(N) - 1$ and thus almost match the this seemingly loose $\log(N)$ bound. Define the "hierarchical" set system $\mathcal{S}_N^{\uparrow}$ over $[N]$ as

$$\mathcal{S}_N^{\uparrow} \coloneqq \{S_1, \ldots, S_N\} \quad \text{where} \quad S_i = \{i, i+1, \ldots, N\} \ .$$

Note that while $\mathcal{S}_N^{\uparrow}$ is defined for all N, it is not asymptotic in the sense discussed in Section 4.1 as the number of groups grows with the number of users N. Further, for this group system the key derivation graphs output by the trivial and asymptotically optimal algorithms coincide, as for every P_I with $P_I \neq \emptyset$ we have $|P_I| = 1$. As the optimal solution for $\mathcal{S}$ is just a path as illustrated in Figure 2, we obtain update costs of

$$\mathrm{Triv}(\mathcal{S}_N^{\uparrow}) = \sum_{i=1}^{N} i \log(i) \approx \frac{N^2}{2}\log(N) \quad \text{and} \quad \mathrm{Opt}(\mathcal{S}_N^{\uparrow}) = \sum_{i=1}^{N} i = \frac{N(N+1)}{2} \approx \frac{N^2}{2} \ ,$$

Thus $\mathrm{Triv}(\mathcal{S}_N^{\uparrow})/\mathrm{Opt}(\mathcal{S}_N^{\uparrow}) \approx \log(N)$ matching the (4) bound. An interesting observation is the fact that an optimal solution can have much larger *depth* than the trivial one: for $\mathcal{S}_N^{\uparrow}$ the depth of the optimal solution is N, while in the trivial solution it's just $\log(N)$. The discussion above indicates that neither the trivial nor the asymptotically optimal algorithm are well-equipped to handle certain group systems. In the following section we propose an algorithm that is not only asymptotically optimal but also generates key-derivation graphs better reflecting the hierarchical nature of group systems, and in particular for the example above recovers the optimal solution.

5 A Greedy Algorithm Based on Huffman Codes

In this section we propose an algorithm to compute key-derivation graphs for group systems. Its formal description is in Section 5.1. In Section 5.2 we compute bounds on its total update cost and compare it to the trivial algorithm and the asymptotically optimal algorithm of Section 4.1 and in Section 5.3 we compute bounds on its worst-case update cost. Finally, in Section 5.4 we show that the algorithm as well as class generalizing it are asymptotically optimal.

5.1 Algorithm Description

We now describe Algorithm 1 that on input of parties $[N]$ and set of groups $\mathcal{S} \subseteq 2^{[N]}$ constructs a key-derivation graph. Its formal description is in Figure 3.

Conceptually, the algorithm proceeds in two phases. The first phase (lines 1 to 11) determines the macro structure of the key-derivation graph. For reasons explained below we will refer to the graph generated in this phase as the *lattice graph*. In the second phase (lines 12 to 20) sources for the individual users are added at the correct position in the lattice graph, which afterwards is binarized to reduce the update size.

More precisely, at the beginning of the first phase the algorithm initializes a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consisting of isolated nodes $v_{S'}$ with $S' \in \mathcal{S}$ that, looking ahead, will represent the group keys. Every node $v_{S'}$ is associated to a set $\mathsf{S}(v_{S'})$ that is initialized to group S'. The algorithm then determines nodes v_1, v_2 such that the intersection of their associated sets is maximal and adds a node v_3 as well as the edges $(v_3, v_1), (v_3, v_2)$ to the graph. The associated set of v_3 is set to $\mathsf{S}(v_1) \cap \mathsf{S}(v_2)$ and the associated sets of v_1 and v_2 are updated to $\mathsf{S}(v_1) \setminus \mathsf{S}(v_3)$ and $\mathsf{S}(v_2) \setminus \mathsf{S}(v_3)$ respectively. This step is repeated until the associated sets of all nodes are pairwise disjoint.

Let $\mathcal{G}_{\mathrm{lat}} = (\mathcal{V}_{\mathrm{lat}}, \mathcal{E}_{\mathrm{lat}})$ denote the resulting lattice graph. In the second phase for every node $v \in \mathcal{V}_{\mathrm{lat}}$ for all $n \in \mathsf{S}(v)$ a source v_n representing user n together with edge (v_n, v') is added to the graph. Finally, for every node v with $\mathrm{indeg}(v) \geq 3$ a Huffman tree from the parents to the node is built. Here the weight of a source is 1, and the weight of non-sources is given as the number of sources below it.

Properties of the lattice graph. We now derive several properties of the lattice graph, which will be used to prove correctness and compute bounds on the total update cost of the generated key-derivation graph. Thus, let $\mathcal{G}_{\mathrm{lat}} = (\mathcal{V}_{\mathrm{lat}}, \mathcal{E}_{\mathrm{lat}})$ be the lattice graph generated on input of $[N]$ and set of k groups $\mathcal{S} = \{S_1, \ldots, S_k\} \subseteq 2^{[N]}$. For index set $I' \subseteq [k]$ we denoted by

$$P_{I'} := \bigcap_{i \in I'} S_i \setminus \bigcup_{j \in [k] \setminus I'} S_j \ ,$$

the set of parties that are members of exactly the groups specified by I. Further, for $v \in \mathcal{V}_{\mathrm{lat}}$ we define

$$I(v) := \{i \in [k] \mid \text{exists path from } v \text{ to } v_{S_i}\} \ ,$$

Input: $(N, \mathcal{S})$

```
G = (V, E) ← (∅, ∅)
for S' ∈ S
  V ← V ∪ {v_{S'}}
  S(v_{S'}) ← S'
while the sets associated to V are not disjoint
  v_1, v_2 ← arg max_{v_1, v_2 ∈ V} (|S(v_1) ∩ S(v_2)|)
  add the node v_3
  S(v_3) ← S(v_1) ∩ S(v_2)
  S(v_1) ← S(v_1) \ S(v_3)
  S(v_2) ← S(v_2) \ S(v_3)
  add the edges (v_3, v_1), (v_3, v_2)
for v ∈ V
  for n ∈ S(v)
    add the node v_n
    add the edge (v_n, v)
    S(v) ← S(v) \ {n}
compute the weight of each node as the number of sources below it
for every node with indegree > 1
  build a Huffman tree from the parents to the node
return G
```

Figure 3: Algorithm 1

the index set of group nodes that can be reached from v. Finally, for a collection $\mathcal{V}' \subseteq \mathcal{V}$ of nodes we generalize the notation for associated sets to $\mathsf{S}(\mathcal{V}') := \cup_{v \in \mathcal{V}'} \mathsf{S}(v)$. We obtain the following.

Lemma 4. *Let $N, k \in \mathbb{N}$, $\mathcal{S} = \{S_1, \ldots, S_k\} \subseteq 2^{[N]}$, and let $\mathcal{G}_{\text{lat}} = (\mathcal{V}_{\text{lat}}, \mathcal{E}_{\text{lat}})$ be the lattice graph generated on input $([N], \mathcal{S})$. Then the following holds.*

1. *Let $v, v' \in \mathcal{V}_{\text{lat}}$ be such that $I(v) = I(v')$. Then $v = v'$.*
2. *$I(v) \neq \emptyset$ for all $v \in \mathcal{V}_{\text{lat}}$.*
3. *For every $v \in \mathcal{V}_{\text{lat}}$ and every $i \in I(v)$ there is exactly one path from v to v_{S_i}.*
4. *Consider the ancestor graph $\mathcal{A}(v)$ for $v \in \mathcal{V}_{\text{lat}}$. Then*
$$\bigcup_{v' \in \mathcal{A}(v)} \mathsf{S}(v') \subseteq \bigcap_{i \in I(v)} S_i \ .$$
If $|I(v)| = 1$ then the equation holds with equality, i.e., $\bigcup_{v' \in \mathcal{A}(v_S)} \mathsf{S}(v') = S$ for all $S \in \mathcal{S}$.
5. *Consider some $v \in \mathcal{V}_{\text{lat}}$. Then we have $\mathsf{S}(v) = P_{I(v)}$.*

Before turning to the proof, we briefly discuss how Lemma 4 allows us to interpret the lattice graph as a subgraph of the Boolean lattice with respect to the power set of $[k]$, i.e., the graph $\mathcal{G}_B = (\mathcal{V}_B, \mathcal{E}_B)$ with $\mathcal{V}_B = \{v_I \mid I \subseteq [k]\}$ and edges $\mathcal{E}_B = \{(v_I, v_{I'}) \mid I, I' \subseteq [k] : I' \subseteq I)\}$. Indeed, Properties 1 and 2 allow us to map every $v \in \mathcal{V}_{\text{lat}}$ to a unique index set $I \subseteq [k]$. Since the existence of an edge $(v, v') \in \mathcal{E}_{\text{lat}}$ implies that $I(v) \supseteq I(v')$ all edges adhere to the structure of $\mathcal{G}_B$. Summing up, the map $\mathcal{G} \to \mathcal{G}_B; v \mapsto v_{I(v)}$ is an injective

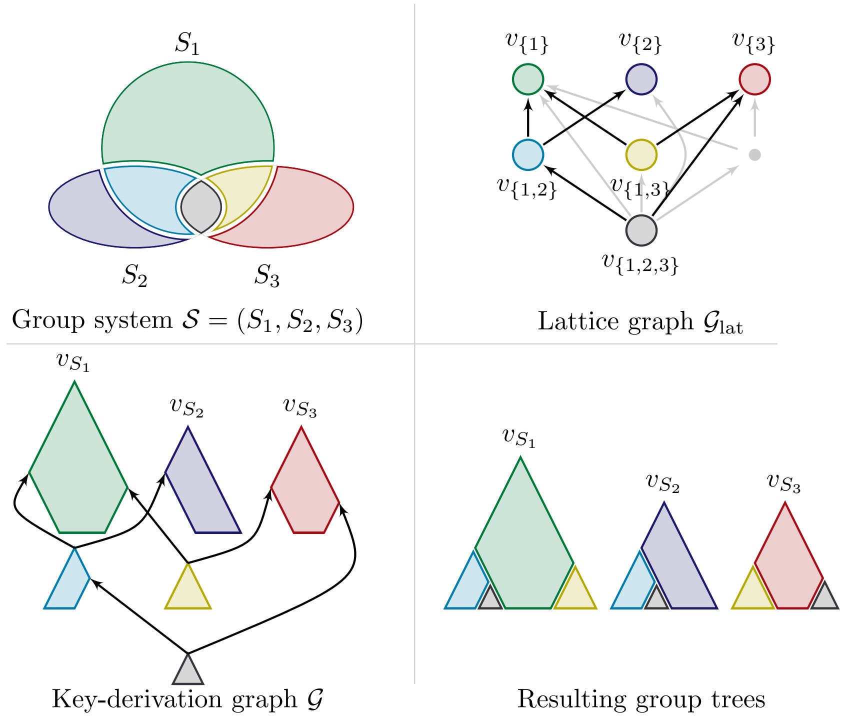

Figure 4: Working principle of the algorithm. Top left; Venn diagram of the considered group system. Top right; resulting lattice graph after the first phase. Node v_I has associated set $\mathsf{S}(v_I) = P_I$, the set of users in exactly the groups indicated by I. Nodes and edges of the Boolean lattice that are not part of $\mathcal{G}_{\text{lat}}$ are depicted in gray. Bottom left; final key derivation graph. Bottom right; resulting trees corresponding to groups S_1, S_2, S_3. Note that components of the same color are shared among different trees.

graph homomorphism. This allows us to identify nodes of the lattice graph with nodes of $\mathcal{G}_B$ and sometimes write $v_{I'}$ for a unique node $v \in \mathcal{V}_{\text{lat}}$ with $I(v) = I' \in \mathcal{P}([k])$. By Property 5 the associated set of v is P_I, the set of users exactly in the groups specified by I. Figure 4 depicts an example execution of Algorithm 1.

Proof of Lemma 4. For $t \in \mathbb{N}$ let $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ be the graph after the tth execution of the loop **while** loop of line 5. We will further use S_t and $\mathcal{A}_t$ to denote associated sets and ancestor sets with respect to $\mathcal{G}_t$.

We show via induction on t that Properties 2 to 4 hold in $\mathcal{G}_t$ and that additionally for all $v \in \mathcal{V}$ the sets associated to $\mathcal{A}_t(v)$ are pairwise disjoint. For $t = 0$ the graph consists of isolated nodes $v_{S_1}, \ldots, v_{S_k}$ with associated sets $S_1, \ldots, S_k$ and corresponding index sets $\{1\}, \ldots, \{k\}$. Thus, all properties stated above clearly hold.

Assume that the properties hold for all steps up to $t-1$ and consider $\mathcal{G}_t$. The only change to $\mathcal{E}$ in the tth step is the addition of two new edges (v_3, v_1) and (v_3, v_2). Thus, by the induction hypothesis we have $I(v) \neq \emptyset$ for all $v \in \mathcal{V}_t \setminus \{v_3\}$. Further $I(v_3) = I(v_1) \cup I(v_2) \neq \emptyset$ and Property 2 still holds.

Regarding Property 3 note that the new node v_3 added in the tth execution of the loop is a source. Thus, by the induction hypothesis for all $v \neq v_3$ there still exists exactly one path to every corresponding v_{S_i}. We have $I(v_3) = I(v_1) \cup I(v_2)$. Since v_3 is connected to both v_1 and v_2, by the induction hypothesis there is at least one path from v_3 to v_{S_i} for all $i \in I(v_3)$. Now assume that there is an i such that there are at least two different paths from v_3 to v_{S_i}. By the induction hypothesis these paths must diverge in v_3, i.e., there are paths from v_1 to v_{s_i} and v_2 to v_{S_i}. But since v_1 and v_2 were chosen by the algorithm in the tth execution of the loop this implies that after the $(t-1)$th execution there were two ancestors of v_{S_i} with non-disjoint associated sets. A contradiction to the induction hypothesis. Thus Property 3 holds.

Regarding Property 4 consider $v \in \mathcal{V}_t$. Note that we either have $\mathcal{A}_t(v) = \mathcal{A}_{t-1}(v)$ or $\mathcal{A}_t(v) = \mathcal{A}_{t-1}(v) \cup \{v_3\}$, v_3 being the newly added node. In the former case Property 4 and disjointness of associated ancestor sets follow immediately from the induction hypothesis. For the latter, first consider the case $v \neq v_3$. Note that the index set $I(v)$ of v remains unchanged. Assume without loss of generality that for the two nodes v_1, v_2 processed by the algorithm in the tth execution of the loop we have $v_1 \in \mathcal{A}_{t-1}(v)$. Then the associated set of v_1 is updated to $\mathsf{S}_t(v_1) = \mathsf{S}_{t-1}(v_1) \setminus \mathsf{S}_{t-1}(v_2)$, $\mathsf{S}_t(v_3) = \mathsf{S}_{t-1}(v_1) \cap \mathsf{S}_{t-1}(v_2)$, and all other associated sets stay unchanged. Thus all sets are still pairwise disjoint and cover the same subset of $\bigcap_{i \in I(v)} S_i$. In particular if $|I(v)| = 1$ we still have $\bigcup_{v \in \mathcal{A}(v)} \mathsf{S}(v) = \bigcap_{i \in I(v)} S_i$. Now consider v_3. We have $I(v_3) = I(v_1) \cup I(v_2)$ which in particular implies $|I(v_3)| \geq 2$. By the induction hypothesis we obtain that

$$\mathsf{S}_t(v_3) = \mathsf{S}_{t-1}(v_1) \cap \mathsf{S}_{t-1}(v_2) \subseteq (\bigcap_{i \in I(v_1)} S_i) \cap (\bigcap_{i \in I(v_2)} S_i) = \bigcap_{i \in I(v_3)} S_i \ .$$

This concludes the induction.

We now prove Property 1. Note that $|I(v)| = 1$ implies $v = v_{S_i}$ for some index i. In this case the property holds by construction. Thus let $v, v' \in \mathcal{V}_{\text{lat}}$ be such that $I(v) = I(v')$ and $|I(v)| \geq 2$. Consider the sets $\mathcal{D}(v)$ and $\mathcal{D}(v')$ of descendants of v and v' respectively. We first show that either $v \in \mathcal{D}(v')$ or $v' \in \mathcal{D}(v)$. To this end, note that by Property 3 the set $M := \{v_{S_i} \mid i \in I(v)\}$ is a subset of $\mathcal{D}(v) \cap \mathcal{D}(v')$. Let $v_1, v_2 \in M$ be such that v_3 was the first node of $\mathcal{D}(v) \cup \mathcal{D}(v')$ added by the algorithm together with the corresponding edges $(v_3, v_1), (v_3, v_2)$, and let t be the point in time when v_3 was added.

Assume without loss of generality that $v_3 \in \mathcal{D}(v)$. We show that $v_3 \in \mathcal{D}(v')$ holds as well. Since $I(v) = I(v')$ there must exist paths from v' to v_1 and v_2. Let v_1^* and v_2^* be the parents of v_1 and v_2 on those paths respectively, and let t_1 and t_2 denote the points in time when v_1^* and v_2^* where added to the graph. Note that by choice of v_3 all nodes that were added before v_3 must have a path to some v_{S_i} with $i \notin I(v)$ and hence by Property 3 cannot be elements of $\mathcal{D}(v')$, which implies that both $t_1 > t$ and $t_2 > t$. If $v_1^* \neq v_3 \neq v_2^*$ then we have

$$\mathsf{S}_{t_1}(v_1^*) \subseteq \mathsf{S}_{t-1}(v_1) \setminus \mathsf{S}_{t-1}(v_2) \quad \text{and} \quad \mathsf{S}_{t_2}(v_2^*) \subseteq \mathsf{S}_{t-1}(v_2) \setminus \mathsf{S}_{t-1}(v_1)$$

since v_1^* and v_2^* were added to the graph after v_3. Thus, the nodes' associated sets are disjoint which excludes the possibility of $t_1^* = t_2^*$ and $v_1^* = v_2^*$. Further, the associated sets of v_1^* and v_2^* being disjoint at the time of their creation contradicts that the nodes share an ancestor v' in the lattice graph. we conclude that the paths must go via v_3 and obtain $v_3 \in \mathcal{D}(v')$.

Note that by Property 3 v_3 is the only node in $\mathcal{D}(v)$ and $\mathcal{D}(v')$ that has edges to v_1 or v_2. By replacing M with $(M \cup \{v_3\}) \setminus \{v_1, v_2\}$ we can use the same argument to show that also the node of $\mathcal{D}(v) \cup \mathcal{D}(v')$ added second must be an element of both $\mathcal{D}(v')$ and $\mathcal{D}(v')$. After finitely many steps we either obtain $v \in \mathcal{D}(v')$ or $v' \in \mathcal{D}(v)$.

Assume without loss of generality the former holds, and assume $v \neq v'$. Then there exists a path from v' to v that uses one of the two outgoing edges of v'. Further, the second outgoing edge must be part of a path to some v_{S_i}, where by correctness $i \in I(v')$. Since $I(v) = I(v')$ there also must exist a path from v to v_{S_i}. Thus, there are at least two paths from v' to v_{S_i} contradicting Property 3. We obtain $v = v'$, which concludes the proof of Property 1.

We now prove Property 5. Consider $v \in \mathcal{V}_{\text{lat}}$. As $v \in \mathcal{A}(v)$ we have by Property 4 that $\mathsf{S}(v) \subseteq \bigcap_{i \in I(v)} S_i$. Assume that there is $n \in \mathsf{S}(v)$ such that $n \in S_j$ for some $j \in [k] \setminus I(v)$. By Property 4, we have that $S_j = \bigcup_{v' \in \mathcal{A}(v_{S_j})} \mathsf{S}(v')$ which would imply that $v \in \mathcal{A}(v_{S_j})$. This however contradicts $j \notin I(v)$ and we obtain

$$\mathsf{S}(v_{I'}) \subseteq \bigcap_{i \in I(v)} S_i \setminus \bigcup_{j \in [k] \setminus I(v)} S_j = P_{I(v)} \ .$$

For the other direction consider $n \in P_{I(v)}$ and let $v' \in \mathcal{V}$ be the node such that $n \in \mathsf{S}(v')$. By Property 4, we have that $n \in \bigcup_{u \in \mathcal{A}(v_{S_i})} \mathsf{S}(u)$ for all $i \in I(v)$ and $n \notin \bigcup_{u \in \mathcal{A}(v_{S_j})} \mathsf{S}(u)$ for all $j \in [k] \setminus I(v)$. This implies that $I(v') = I(v)$. By Property 1 we obtain $v = v'$ and in turn $P_{I(v)} \supseteq \mathsf{S}(v)$. $\square$

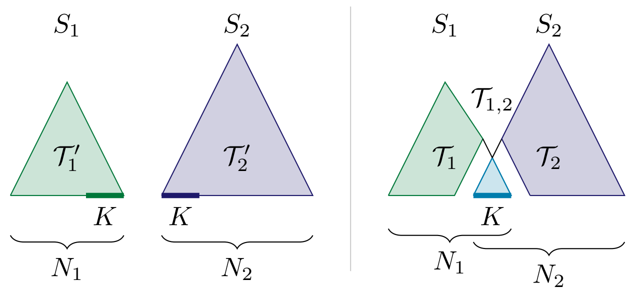

Figure 5: Key-derivation graphs of the trivial algorithm (left) and Algorithm 1 (right) for two subgroups. Users that are members of both subgroups are marked in thick.

Correctness. We show that key-derivation graph $\mathcal{G}$ output by Algorithm 1 satisfies the correctness properties of Definition 1. Note that the first property holds by construction.

To see that the second property holds as well, consider the lattice graph. By Lemma 4, Property 4 for every group $S' \in \mathcal{S}$ the associated sets of the ancestors of $v_{S'}$ form a partition of S'. In the second phase of the algorithm a source v_n is added for every user and connected to corresponding node in the lattice graph. Thus, after this step the set of users with a path to $v_{S'}$ is exactly S'. As this property remains unaffected by the binarization step of line 19 the final key-derivation graph is indeed correct.

5.2 Total Update Cost

In this Section we analyze the total update cost $\text{Upd}(\mathcal{G}) = \sum_{n\in[N]} \text{Upd}(n)$ of key-derivation graphs $\mathcal{G}$ generated by Algorithm 1. To this end, we will split $\text{Upd}(\mathcal{G})$ into the contribution made by the constituting Huffman trees $\mathcal{T}$. Tree $\mathcal{T}$ has a single root and all non-sources in $\mathcal{T}$ have indegree 2. Let $\mathcal{L}(\mathcal{T})$ denote the set of leaves of $\mathcal{T}$. As argued in Lemma 2, the update cost of a leaf u with respect to $\mathcal{T}$ corresponds to the length $\text{len}(u)$ of its path to the root. Note, however, that leaves of $\mathcal{T}$ may represent more than one user in the key-derivation graph. Indeed, by construction of the algorithm the weight w_u of u counts the number of leaves in $\mathcal{G}$ below u. Thus, the contribution of Huffman tree $\mathcal{T}$ towards the total update cost of $\mathcal{G}$ is given by $\text{Upd}(\mathcal{T}) = \sum_{u\in\mathcal{L}(\mathcal{T})} w_u \text{len}(u)$. If $U_{\mathcal{T}}$ is the probability distribution that picks $u \in \mathcal{L}(\mathcal{T})$ with probability proportional to its weight w_u we can express the update cost of $\mathcal{T}$ in terms of the expected length from leaves to the root as

$$\text{Upd}(\mathcal{T}) = \mathbb{E}[\text{len}(U_{\mathcal{T}})] \cdot \sum_{u\in\mathcal{L}(\mathcal{T})} w_u \quad . \tag{6}$$

We first consider Algorithm 1 for the simplest case of two subgroups and compare it to the trivial algorithm.

Example 1. *Let $N \in \mathbb{N}$ and let $\mathcal{S}$ consist of two subgroups S_1, S_2 of sizes N_1 and N_2 respectively. Further assume that $|S_1 \cap S_2| = K$. Consider the key derivation graphs generated by the trivial algorithm and Algorithm 1, which in both cases decompose into several Huffman trees. The trivial algorithm essentially generates two trees $\mathcal{T}_1'$ and $\mathcal{T}_2'$, the first containing all members of S_1, the other all members of S_2. Algorithm 1 first collects the K parties that are members of both groups in a tree $\mathcal{T}_{1,2}$. The remaining $(N_1 - K)$ members of S_1 and the root of $\mathcal{T}_{1,2}$ are collected in a tree $\mathcal{T}_1$, the remaining $(N_2 - K)$ members of S_2 and the root of $\mathcal{T}_{1,2}$ in a tree $\mathcal{T}_2$ (See Figure 5).*

By Equation 6 we have

$$\text{Upd}(\mathcal{G}_{\text{triv}}) = \text{Upd}(\mathcal{T}_1') + \text{Upd}(\mathcal{T}_2') = N_1\, \mathbb{E}[\text{len}(U_{\mathcal{T}_1'})] + N_2\, \mathbb{E}[\text{len}(U_{\mathcal{T}_2'})]$$

and

$$\begin{aligned}\text{Upd}(\mathcal{G}_{\text{a1}}) &= \text{Upd}(\mathcal{T}_1) + \text{Upd}(\mathcal{T}_2) + \text{Upd}(\mathcal{T}_{1,2}) \\ &= N_1\, \mathbb{E}[\text{len}(U_{\mathcal{T}_1})] + N_2\, \mathbb{E}[\text{len}(U_{\mathcal{T}_2})] + K\, \mathbb{E}[\text{len}(U_{\mathcal{T}_{1,2}})] \quad .\end{aligned}$$

By optimality of Huffman codes (Lemma 1) we have that

$$H(U_{\mathcal{T}}) \le \mathbb{E}[\mathrm{len}(U_{\mathcal{T}})] \le H(U_{\mathcal{T}}) + 1$$

for $\mathcal{T} \in \{\mathcal{T}_1', \mathcal{T}_2', \mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_{1,2}\}$*, where* $H(U_{\mathcal{T}})$ *is the Shannon entropy of* $U_{\mathcal{T}}$*. For* $\mathcal{T}_1'$*,* $\mathcal{T}_2'$*, and* $\mathcal{T}_{1,2}$ *the leaves are distributed uniformly and we have* $H(\mathcal{T}_1') = \log(N_1)$*,* $H(\mathcal{T}_2') = \log(N_2)$*,* $H(\mathcal{T}_{1,2}) = \log(K)$*. Let* $i \in \{1,2\}$ *and consider* $\mathcal{T}_i$*. Then the first* $N_i - K$ *leaves have probability* $1/N_i$ *and the last leaf* K/N_i*. Thus* $H(U_{\mathcal{T}_i}) = (N_i - K)/N_i \log(N_i) + K/N_i \log(N_i/K) = \log(N_i) - K/N_i \log(K)$*. Summing up we obtain*

$$\begin{aligned}
&\mathrm{Upd}(\mathcal{G}_{\mathrm{triv}}) - \mathrm{Upd}(\mathcal{G}_{\mathrm{a1}}) \\
\ge& N_1 \log(N_1) + N_2 \log N_2 - N_1(\log(N_1) - K/N_1 \log(K) + 1) \\
&\quad - N_2(\log(N_2) - K/N_2 \log(K) + 1) - K(\log(K) + 1) \\
=& K(\log(K) - 1) - (N_1 + N_2) \ .
\end{aligned}$$

Note that for $K \ge 2$ *the first term is non-negative (For* $K = 1$ *it is easy to see that Algorithm* 1 *performs better than the trivial algorithm.).*

Before turning to arbitrary group systems we derive a generalized statement on the update cost $\mathrm{Upd}(\mathcal{T})$ contributed by Huffman trees as defined above.

Lemma 5. *Let* $\mathcal{T}$ *be a Huffman tree over leaves* $v_1, \dots, v_\ell$ *of weight* $w_1, \dots, w_\ell \in \mathbb{N}$*. Let* $w = \sum_{i=1}^{\ell} w_i$*. Then* $\mathcal{T}$*'s update cost is bounded by*

$$w \log(w) - \sum_{i=1}^{\ell} w_i \log(w_i) \le \mathrm{Upd}(\mathcal{T}) \le w(\log(w) + 1) - \sum_{i=1}^{\ell} w_i \log(w_i) \ .$$

Proof. Let $U_{\mathcal{T}}$ denote the probability distribution that picks leaf v_i with probability w_i/w proportional to its weight. The entropy of $U_{\mathcal{T}}$ is given by

$$\begin{aligned}
H(U_{\mathcal{T}}) &= -\sum_{i=1}^{\ell} \frac{w_i}{w} \log(w_i/w) = \sum_{i=1}^{\ell} \frac{w_i}{w} \log(w) - \sum_{i=1}^{\ell} \frac{w_i}{w} \log(w_i) \\
&= \log(w) - \sum_{i=1}^{\ell} \frac{w_i}{w} \log(w_i) \ .
\end{aligned}$$

The claim follows since by Equation 6 the update cost of $\mathcal{T}$ with respect to the weights is given by $\mathbb{E}[\mathrm{len}(U_{\mathcal{T}})] \cdot w$ and by optimality of Huffman codes $H(U_{\mathcal{T}}) \le \mathbb{E}[\mathrm{len}(U_{\mathcal{T}})] \le H(U_{\mathcal{T}}) + 1$. □

Regarding general systems of subgroups we obtain the following.

Theorem 1. *Let* $N \in \mathbb{N}$*,* $S_1, \dots, S_k \subseteq [N]$*, and* $\mathcal{G}$ *the key-derivation graph output by Algorithm* 1*. Let* $\mathcal{G}_{\mathrm{lat}} = (\mathcal{V}_{\mathrm{lat}}, \mathcal{E}_{\mathrm{lat}})$ *be the corresponding lattice graph. Then*

$$\begin{aligned}
&\sum_{i=1}^{k} |S_i| \cdot \log(|S_i|) - \sum_{v \in \mathcal{V}_{\mathrm{lat}} \colon |I(v)| \ge 2} \Big| \bigcup_{v' \in \mathcal{A}(v)} P_{I(v')} \Big| \cdot \log\Big(\Big| \bigcup_{v' \in \mathcal{A}(v)} P_{I(v')} \Big| \Big) && (7) \\
\le& \mathrm{Upd}(\mathcal{G}) \\
\le& \sum_{i=1}^{k} |S_i| \cdot (\log(|S_i|) + 1) - \sum_{v \in \mathcal{V}_{\mathrm{lat}} \colon |I(v)| \ge 2} \Big| \bigcup_{v' \in \mathcal{A}(v)} P_{I(v')} \Big| \cdot \Big(\log\Big(\Big| \bigcup_{v' \in \mathcal{A}(v)} P_{I(v')} \Big| \Big) - 1 \Big) \ , && (8)
\end{aligned}$$

where $\mathcal{A}(v)$ *denotes the set of ancestors of* v *in* $\mathcal{G}_{\mathrm{lat}}$*,* $I(v) = \{i \in [k] \colon \exists \text{ path from } v \text{ to } v_{S_i}\}$*, and for* $I' \subseteq [N]$ *the set* $P_{I'} \coloneqq \bigcap_{i \in I'} S_i \setminus \bigcup_{j \in [k] \setminus I'} S_j$ *indicates the users exactly in the subgroup corresponding to* I'*.*

Proof. As in Example 1 we decompose the total update cost of $\mathcal{G}$ into parts contributed by Huffman trees. Note that every node $v' \in \mathcal{V}_{\text{lat}}$ of the lattice graph serves as the root of a Huffman tree $\mathcal{T}_{v'}$ in the final key-derivation graph $\mathcal{G}$ and we can compute the total update cost of $\mathcal{G}$ as $\text{Upd}(\mathcal{G}) = \sum_{v' \in \mathcal{V}_{\text{lat}}} \text{Upd}(\mathcal{T}_{v'})$.

The leaves $\mathcal{L}(\mathcal{T}_{v'})$ of $\mathcal{T}_{v'}$ are either sources that were added in the second phase of the algorithm, or nodes that are a parent of v' in the lattice graph.

Recall that the weight of leaves is defined as 1 and for general nodes as the number of leaves below it. This implies that node v'' in line 17 of the algorithm gets assigned weight $|\mathsf{S}(\mathcal{A}(v''))| = \left|\bigcup_{\tilde{v} \in \mathcal{A}(v'')} \mathsf{S}(\tilde{v})\right|$.

Let $v'_1, \dots, v'_\ell$ denote the parents of v' in $\mathcal{G}_{\text{lat}}$ and consider Huffman tree $\mathcal{T}_{v'}$. Then $\mathcal{T}_{v'}$ has leaves v'_i with weight $w_{v'_i} = |\mathsf{S}(\mathcal{A}(v'_i))|$ for $i \in \{1, \dots, \ell\}$, and $|\mathsf{S}(\mathcal{A}(v'))| - \sum_{i=1}^{\ell} |\mathsf{S}(\mathcal{A}(v'_i))|$ additional leaves, each of which has weight 1.

In the following we will use f as shorthand for the function $f\colon n \mapsto n \log(n)$. We now bound $\text{Upd}(\mathcal{T}_{v'})$ using Lemma 5. As the negative terms in Lemma 5's statement contributed by leaves of weight 1 are $1 \cdot \log(1) = 0$ we obtain that

$$\text{Upd}(\mathcal{T}_{v'}) \geq f(|\mathsf{S}(\mathcal{A}(v'))|) - \sum_{i=1}^{\ell} f(|\mathsf{S}(\mathcal{A}(v'_i))|)$$

and

$$\text{Upd}(\mathcal{T}_{v'}) \leq f(|\mathsf{S}(\mathcal{A}(v'))|) + |\mathsf{S}(\mathcal{A}(v'))| - \sum_{i=1}^{\ell} f(|\mathsf{S}(\mathcal{A}(v'_i))|) \ .$$

To compute the total update cost of $\mathcal{G}$ we have to sum over all trees $\mathcal{T}_{v'}$ with $v' \in \mathcal{V}_{\text{lat}}$. Note that every node v' in $\mathcal{V}_{\text{lat}}$ with $|I(v')| \geq 2$ has outdegree 2. Thus if we sum over the update cost of all trees the term $f(|\mathsf{S}(\mathcal{A}(v'))|$ appears twice with a negative sign (in the cost of v''s children) and once with a positive (in $\text{Upd}(\mathcal{T}_{v'})$). Thus

$$\begin{aligned}
&\sum_{v' \in \mathcal{V}_{\text{lat}} \colon |I(v')|=1} f(|\mathsf{S}(\mathcal{A}(v'))|) - \sum_{v' \in \mathcal{V}_{\text{lat}} \colon |I(v')| \geq 2} f(|\mathsf{S}(\mathcal{A}(v'))|) \\
\leq &\text{Upd}(\mathcal{G}) = \sum_{v' \in \mathcal{V}_{\text{lat}}} \text{Upd}(\mathcal{T}_{v'}) \\
\leq &\sum_{v' \in \mathcal{V}_{\text{lat}} \colon |I(v')|=1} \left(f(|\mathsf{S}(\mathcal{A}(v'))|) + |\mathsf{S}(\mathcal{A}(v'))|\right) \\
&\quad - \sum_{v' \in \mathcal{V}_{\text{lat}} \colon |I(v')| \geq 2} \left(f(|\mathsf{S}(\mathcal{A}(v'))|) - |\mathsf{S}(\mathcal{A}(v'))|\right) \ .
\end{aligned}$$

This is equivalent to the claim of the theorem since the nodes v' with $|I(v')| = 1$ are exactly the group-key nodes of the form $v' = v_{S_i}$ and since by correctness of the algorithm $|\mathsf{S}(\mathcal{A}(v_{S_i}))| = |S_i|$ and by Lemma 4, Property 5 $\mathsf{S}(v_{I'}) = P_{I'}$ where the partitions $P_{I'}$ are disjoint. ☐

The bounds of Theorem 1 depend on the structure of the lattice graph generated by the algorithm. Using Properties 4 and 5 of Lemma 4 to bound $|\mathsf{S}(\mathcal{A}(v'))|$ it is possible to obtain a weaker bound on $\text{Upd}(\mathcal{G})$ that only depends on $[N]$ and $\mathcal{S}$.

We conclude the section by comparing the update cost of Algorithm 1 to the trivial algorithm and the asymptotically optimal algorithm of Section 4.1.

Comparison to the trivial algorithm. Note that the terms $\sum_{i=1}^{k} |S_i| \cdot \log(|S_i|)$ and $\sum_{i=1}^{k} |S_i| \cdot (\log(|S_i|) + 1)$ in Theorem 1 match the bounds on the update cost of the trivial algorithm derived in Section 3.4. Thus the second term of

$$\sum_{v \in \mathcal{V}_{\text{lat}} \colon |I(v)| \geq 2} \left| \bigcup_{v' \in \mathcal{A}(v)} P_{I(v')} \right| \cdot \left(\log \left(\left| \bigcup_{v' \in \mathcal{A}(v)} P_{I(v')} \right| \right) - 1 \right)$$

provides a good estimate on how much Algorithm 1 saves compared to the trivial one. For the group system depicted in Figure 4, for example, this would amount to

$$|S_1 \cap S_2| \cdot \log(|S_1 \cap S_2|) + |S_1 \cap S_3 \setminus S_2| \cdot \log(|S_1 \cap S_3 \setminus S_2|) + |S_1 \cap S_2 \setminus S_3| \cdot \log(|S_1 \cap S_2 \cap S_3|) \ .$$

Due to the "rounding error" of +1 in $\sum_{i=1}^{k} |S_i| \cdot (\log(|S_i|) + 1)$ Theorem 1 unfortunately does not allow us to conclude that the update cost of Algorithm 1 always improves on the one of the trivial algorithm. In Appendix A we provide an alternative analysis of $\mathrm{Upd}(\mathcal{G})$ that directly compares the two algorithms and gives conditions that imply Algorithm 1 outperforming the trivial one.

Comparison to the asymptotically optimal algorithm of Section 4.1. The algorithm of Section 4.1 in a first step constructs a binary tree for every non-empty partition $P_{I'}$ and then in a second step for every group builds a binary tree using the roots of the "partition trees" as leafs. We can interpret this as an algorithm that similarly to Algorithm 1 in the first phase chooses a lattice graph $\mathcal{G}_{\text{lat}}$, concretely the graph that connects every node $v_{I'}$ directly with edges to all corresponding group nodes $\{v_{\{i\}} \mid i \in I'\}$, and in the second phase builds Huffman trees for every lattice node.[6]

Thus, by Lemma 5 we can lower bound the update cost of key graphs $\mathcal{G}_{\text{asopt}}$ generated by it by

$$\mathrm{Upd}(\mathcal{G}_{\text{asopt}}) \geq \sum_{i=1}^{k} \Big(|S_i| \cdot \log(|S_i|) - \sum_{I' \subseteq [N]: i \in I' \wedge |I'| \geq 2} |P_{I'}| \cdot \log(|P_{I'}|) \Big) + \sum_{I' \subseteq [N]: |I'| \geq 2} |P_{I'}| \cdot \log(|P_{I'}|) \ ,$$

which, taking into account that every I' with $|I'| = \ell$ corresponds to exactly ℓ groups, simplifies to

$$\mathrm{Upd}(\mathcal{G}_{\text{asopt}}) \geq \sum_{i=1}^{k} |S_i| \cdot \log(|S_i|) - \sum_{I' \subseteq [N]: |I'| \geq 2} (|I'| - 1) \, |P_{I'}| \cdot \log(|P_{I'}|) \ . \tag{9}$$

For a comparison to Algorithm 1, consider a key derivation graph $\mathcal{G}_{\text{a1}}$ output by it. We now compute a lower bound on $\mathrm{Upd}(\mathcal{G}_{\text{asopt}}) - \mathrm{Upd}(\mathcal{G}_{\text{a1}})$. Let $\mathcal{G}'_{\text{lat}}$ be the lattice graph of $\mathcal{G}_{\text{a1}}$ and $v_{I'} \in \mathcal{G}'_{\text{lat}}$ such that $|I'| \geq 2$. Every non-sink in $\mathcal{G}'_{\text{lat}}$ has outdegree 2 and $v_{I'}$ is connected to all $v_{\{i\}}$ with $i \in I'$ by exactly one path. Thus, the subgraph of $\mathcal{G}'_{\text{lat}}$ induced by these paths is a binary tree with root $v_{I'}$ and $|I'|$ leafs, and thus consists of exactly $2\,|I'| - 1$ nodes, $|I'|$ of which have an index set of size 1. This implies that there exists $|I'| - 1$ many nodes $v_{I''}$ in $\mathcal{G}'_{\text{lat}}$ with $|I''| \geq 2$ such that $v_{I'} \in \mathcal{A}(v_{I''})$.

Using f as shorthand for the function $f : N \mapsto N \log(N)$ and $p_{I'} = |P_{I'}|$, we now can distribute the expressions $|P_{I'}| \cdot \log(|P_{I'}|)$ of Equation 9 on the negative summands of Equation 8 and obtain

$$\mathrm{Upd}(\mathcal{G}_{\text{asopt}}) - \mathrm{Upd}(\mathcal{G}_{\text{a1}}) \geq \sum_{v \in \mathcal{V}'_{\text{lat}}:\ |I(v)| \geq 2} \big(f\big(\sum_{v' \in \mathcal{A}(v)} p_{I(v')}\big) - \sum_{v' \in \mathcal{A}(v)} f(p_{I(v')}) - 1\big) \ .$$

Note that the function f grows super-linearly implying that the terms $f(\sum_{v' \in \mathcal{A}(v)} p_{I(v')}) - \sum_{v' \in \mathcal{A}(v)} f(p_{I(v')})$ are non-negative, and can even be of order N as for example $f(2N/2) - 2f(N/2) = N$. While again due to the terms -1 we are unfortunately not able to conclude that Algorithm 1 is always more efficient, this shows that it still can save substantially in terms of update cost, in particular if the $p_{I'}$ are large.

5.3 Maximal Update Cost per User

In the previous section we were considering the total update cost of key-derivation graphs generated by Algorithm 1, which relates to the average update cost of parties. As we have shown this metric will typically improve compared to the trivial algorithm. However, it might still be possible, that the update cost of

[6]Formally, the algorithm as described in Section 4.1 collects all users that are *only* in group S_i in a tree before computing the tree for S_i, while in the lattice-graph variant these users are directly included in the tree for S_i. Note, however, that the latter approach can only improve the total update cost.

particular, fixed users increases. In this section we show that while this may indeed happen, the increase is essentially bounded by a small constant.

As Algorithm 1 builds on Huffman codes, the results of this section make use of weight distributions that maximize the codeword length of such codes, concretely, weights corresponding to the Fibonacci numbers F_i that are recursively defined by

$$F_0 = 0, F_1 = 1 \quad \text{and} \quad F_i = F_{i-1} + F_{i-2} \text{ for } i \geq 2 \ .$$

We will make use of the following facts from [1]:

$$\sum_{i=1}^{k} F_i = F_{k+2} - 1 \tag{10}$$

$$k - 2 < \log_\Phi(F_k) < k - 1 \ , \tag{11}$$

where $\Phi = (1 + \sqrt{5})/2$.

We first consider an example in which the update cost of a fixed user increases compared to the trivial solution.

Example 2. *Recall that for a system of subgroups $\mathcal{S} = \{S_1, \cdots, S_k\}$ the set of parties exactly in the subgroups specified by index set $I \subseteq [k]$ is given by $P_I = \bigcap_{i\in I} S_i \setminus \bigcup_{j\in[k]\setminus I} S_j$. Now assume that $\mathcal{S}$ satisfies*

$$\left|P_{\{1\}}\right| = 1, \quad \left|P_{\{1,i\}}\right| = F_i \ \ \forall i \in \{2, \cdots, k\}, \ \textit{and} \quad |P_I| = 0 \textit{ for all other } I \ .$$

We are interested in the update cost of the single party $n \in P_{\{1\}}$. Since by choice of the P_I we have that $|S_1| = \sum_{i=1}^{k} F_i$, we obtain by Equations 10 and 11 that n's update cost with respect to the trivial algorithm is maximally $\mathrm{Upd}_{\mathrm{triv}}(n) \leq \lceil (k+1)\log(\Phi)\rceil$.

Now consider n's update cost in a key-derivation graph $\mathcal{G}$ generated by our algorithm. Then v_n is a leaf of weight 1 in the Huffman tree with root v_{S_1}, while the other leaves $k-1$ have weights corresponding to the Fibonacci sequence. Thus, the length of v_n's path to the root and in turn her update cost $\mathrm{Upd}(n)$ is $k-1$.

Summing up, for the considered set system the maximal update cost with respect to Algorithm 1 is larger by a factor of roughly $1/\log(\Phi) \approx 1.44$ compared to the one of the trivial algorithm.

Below we show that the behavior exhibited in the example above is essentially the worst case. We first recall a fact about the maximal length of paths in Huffman trees that follows from plugging Equation 11 into [1, Theorem 5].

Fact 1. *Let $\mathcal{T}$ be a Huffman tree over leaves $v_1, \ldots, v_\ell$ of weight $w_1, \ldots, w_\ell \in \mathbb{N}$. Let $w = \sum_{i=1}^{\ell} w_i$. Then for all $j \in \{1, \ldots, \ell\}$ the length of the path from w_j to the root of $\mathcal{T}$ is bounded by*

$$\mathrm{len}(w_j) \leq \log_\Phi(w) - \log_\Phi(w_j) + 1 \ ,$$

where $\Phi = (1 + \sqrt{5})/2$.

Lemma 6. *Let $N \in \mathbb{N}$, $\mathcal{S} = \{S_1, \ldots, S_k\} \subseteq 2^{[N]}$, and $\mathcal{G}$ the key-derivation graph output by Algorithm 1. Fix a party $n \in [N]$ and let $I' =:= \{i \in [k] \mid n \in S_i\}$. Then n's update cost in $\mathcal{G}$ is bounded from above by*

$$\mathrm{Upd}(n) \leq \sum_{i\in I'} \big(\lceil \log_\Phi(|S_i|)\rceil + |I'| - 1 \big) \approx \sum_{i\in I'} \big(\lceil 1.44 \cdot \log(|S_i|)\rceil + |I'| - 1 \big) \ .$$

Proof. Let $\mathcal{G}_{\mathrm{lat}} = (\mathcal{V}_{\mathrm{lat}}, \mathcal{E}_{\mathrm{lat}})$ denote the lattice graph corresponding to $\mathcal{G}$. By Lemma 4, Property 5 the node $v'_0 \in \mathcal{V}_{\mathrm{lat}}$ that the source v_n is connected to in the second phase of the algorithm satisfies $I(v'_0) = I'$. Fix $i \in I'$. By Property 3 v'_0 is connected to group node v_{S_i} by exactly one path $(v'_0, \cdots, v'_\ell = v_{S_i})$. Since $I(v'_0) = I'$ and $I(v'_{j-1}) \supsetneq I(v'_j)$ for all j we have $\ell \leq |I'| - 1$. In the key-derivation graph v'_j is connected to v'_{j-1} by a path P_j for $j \in \{1, \ldots, \ell\}$ and party n's source v_n is connected to v'_0 by a path P_0.

Since the weight of v'_j is $\left|\mathsf{S}(\mathcal{A}(v'_j))\right|$ we obtain by Fact 1 that

$$\operatorname{len}(P_j) \le \left\lceil \log_\Phi(\left|\mathsf{S}(\mathcal{A}(v'_j))\right|) - \log_\Phi(\left|\mathsf{S}(\mathcal{A}(v'_{j-1}))\right|)\right\rceil$$

and that $\operatorname{len}(P_0) \le \lceil \log_\Phi(|\mathsf{S}(\mathcal{A}(v'_0))|)\rceil$ since v_n has weight 1. Using that $\lceil a-b\rceil + \lceil b\rceil \le \lceil a\rceil + 1$ for $b, a-b \ge 0$ it follows that

$$\begin{aligned}\sum_{j=0}^{\ell} \operatorname{len}(P_j) &\le \lceil \log_\Phi(|\mathsf{S}(\mathcal{A}(v'_0))|)\rceil + \sum_{j=1}^{\ell} \left\lceil \log_\Phi(\left|\mathsf{S}(\mathcal{A}(v'_j))\right|) - \log_\Phi(\left|\mathsf{S}(\mathcal{A}(v'_{j-1}))\right|)\right\rceil \\ &\le \lceil \log_\Phi(|\mathsf{S}(\mathcal{A}(v_\ell))|)\rceil + \ell \\ &\le \lceil \log_\Phi(|\mathsf{S}(\mathcal{A}(v_\ell))|)\rceil + |I'| - 1 \ .\end{aligned}$$

Summing over all $i \in I'$ and taking into account that by Lemma 4, Property 4 $\mathsf{S}(\mathcal{A}(v_{S_i})) = S_i$ yields the claim of the lemma.

□

Note that in the analysis above the component of n's update cost contributed by the Huffman tree rooted at lattice node v_I is included in the bound of the update cost of all S_i with $i \in I$, and thus overestimated by a factor of $|I|$. Thus, the actual update cost of users (in particular if they are members of many groups) will typically be better than Lemma 6 indicates.

5.4 Asymptotic Optimality of Boolean-lattice based Graphs

As discussed in Section 5.1 we can interpret our algorithm as follows. On input $([N], \mathcal{S} = \{S_1, \dots, S_k\})$ in the first phase the algorithm picks a subgraph of the Boolean lattice $\mathcal{G}_B = (\mathcal{V}_B, \mathcal{E}_B)$ with respect to the power set of $[k]$, where

$$\mathcal{V}_B = \{v_I \mid I \subseteq [k]\} \quad \text{and} \quad \mathcal{E}_B = \{(v_I, v_{I'}) \mid I, I' \subseteq [k] : I' \subseteq I)\} \ .$$

We refer to this subgraph as the lattice graph. In the second phase for $I \subseteq [k]$ a source for every party in P_I, i.e., the set of parties belonging exactly to the groups specified by I, is added and connected to node v_I. Each node in the graph is assigned a weight; sources have weight 1 and the weight of all other nodes is the sum of the weights of their parents. Finally, for every v_I a Huffman tree to its parents according to the weight distribution is built, resulting in the key-derivation graph.

In this section we consider key-derivation graphs for general choices of the lattice graph, i.e., key derivation graphs $\mathcal{G}$ obtained by executing the second phase of the algorithm as described above with respect to a lattice-graph $\mathcal{G}_{\text{lat}} = (\mathcal{V}_{\text{lat}}, \mathcal{E}_{\text{lat}}) \subseteq \mathcal{G}_B$.[7] We say $\mathcal{G}$ is the key-derivation graph associated to $\mathcal{G}_{\text{lat}}$, $[N]$ and $\mathcal{S}$. The following theorem shows that the update cost of every lattice-based key derivation graphs, and in particular graphs generated by Algorithm 1, is optimal in the asymptotic setting of Section 4.

Theorem 2. *Let $k \in \mathbb{N}$ be fixed, and for $I \subseteq [k]$ let $p_I \in [0,1]$ be such that $\sum_{I \subseteq [k]} p_I = 1$ and $p_\emptyset = 0$. For $N \in \mathbb{N}$ let $\mathcal{S}(N)$ be the subgroup system associated to the p_I.*

Let $\mathcal{G}_{\text{lat}} = (\mathcal{V}_{\text{lat}}, \mathcal{E}_{\text{lat}})$ be a subgraph of the Boolean-lattice graph with respect to $[k]$ satisfying that $v_I \in \mathcal{V}_{\text{lat}}$ for all I with $p_I > 0$, and let $\mathcal{G}(N)$ be the key-derivation graph associated to $\mathcal{G}_{\text{lat}}$ and $\mathcal{S}(N)$. Then

$$\operatorname{Upd}(\mathcal{G}(N)) \xrightarrow{N\to\infty} \sum_{I \subseteq [k]} |N \cdot p_I| \cdot \log(|N \cdot p_I|) + \Theta(N) = N \log(N) + \Theta(N) \ .$$

[7]Naturally, one would require that the resulting key-derivation graph satisfies correctness. However, this is not necessary for our analysis of its update cost.

Proof. For node $v_I \in \mathcal{V}_{\text{lat}}$ let $\mathcal{T}_{v_I}^N$ denote the Huffman-tree in $\mathcal{G}(N)$ rooted at v_I. Further, let $v_1, \cdots, v_\ell$ be the parents of v_I in $\mathcal{G}_{\text{lat}}$. We will analyze the contribution $\mathcal{T}_{v_I}^N$ makes to the total update cost of $\mathcal{G}(N)$.

First, we show that if $p_I > 0$ then

$$\text{Upd}(\mathcal{T}_{v_I}^N) \xrightarrow{N\to\infty} Np_I \cdot \log(Np_I) \ .$$

To this end, for $i \in \{1, \ldots, \ell\}$ let q_i be such that the weight of v_i in $\mathcal{G}(N)$ is given by $q_i N$. Since $\mathcal{T}_{v_I}^N$ has $p_I \cdot N$ leaves of weight 1 additional to $v_1, \ldots, v_\ell$ this implies that the weight of v_I in $\mathcal{G}(N)$ is $N \cdot (p_I + \sum_{i=1}^{\ell} q_i)$. We now bound $\text{Upd}(\mathcal{T}_v^N)$ using Lemma 5. As the negative terms contributed by leaves of weight 1 are $1 \cdot \log(1) = 0$ we obtain that

$$\begin{aligned}
&\text{Upd}(\mathcal{T}_{v_I}^N) \\
\leq& N\Big((p_I + \sum_{i=1}^{\ell} q_i) \cdot \log\big(N(p + \sum_{i=1}^{\ell} q_i)\big) - \sum_{i=1}^{\ell} q_i \log(N \cdot q_i)\Big) \\
=& N\Big(p_I \log(N) + \sum_{i=1}^{\ell} q_i \log(N) + p_I \log(p_I + \sum_{i=1}^{\ell} q_i) + \sum_{i=1}^{\ell} q_i \log(p_I + \sum_{i=1}^{\ell} q_i) \\
&\quad - \sum_{i=1}^{\ell} q_i \log(N) - \sum_{i=1}^{\ell} q_i \log(q_i)\Big) \ ,
\end{aligned}$$

Note that the terms $\sum_{i=1}^{\ell} q_i \log(N)$ cancel out and that $c := p_I \log(p_I + \sum_{i=1}^{\ell} q_i) + \sum_{i=1}^{\ell} q_i \log(p_I + \sum_{i=1}^{\ell} q_i) - \sum_{i=1}^{\ell} q_i \log(q_i)$ is independent of N. We thus have

$$\text{Upd}(\mathcal{T}_{v_I}^N) \leq N \cdot (p_I \log(N) + c)$$

and obtain in the case $p > 0$ that

$$1 \leq \frac{\text{Upd}(\mathcal{T}_{v_I})}{Np_I \cdot \log(Np_I)} \leq \frac{p_I \log(N) + c}{p_I \log(N) + p_I \log(p_I)} \ ,$$

where the first inequality is due to Lemma 5 and the last term converges to 1 as claimed.

Now consider the case $p_I = 0$. In this case the Huffman tree $\mathcal{T}_{v_I}^N$ for all N has exactly ℓ leaves, the proportional weight of which stays unchanged. Thus $\mathcal{T}_{v_I}^N$ is the same for all N and in particular has constant average update size. Recall that by Equation 6 the update cost of $\mathcal{T}_{v_I}^N$ is given by

$$\text{Upd}(\mathcal{T}_{v_I}^N) = \mathbb{E}[\text{len}(U_{\mathcal{T}_{v_I}^N})] \cdot w_{v_I} \ .$$

Since $\mathbb{E}[\text{len}(U_{\mathcal{T}_{v_I}^N})]$ as argued above is constant, and since all weights w_{v_I} are linear in N we obtain that $\text{Upd}(\mathcal{T}_{v_I}^N) \in O(N)$.

Summing over all Huffman trees yields the claim of the theorem. ☐

6 Dynamic Operations

So far we considered the setting of systems $\mathcal{S}$ of *static* groups for a universe of users $[N]$, i.e., while the keys in the key-derivation graph are rotated, the set of groups that a particular party is a member of stays unchanged. Naturally, we would like to be able to add or remove users from groups. In this section, we first analyze what these operations correspond to with respect to boolean lattice based key-derivation graphs and then discuss how techniques for adds and removes in CGKA and Multicast schemes in the single-group setting can be adapted to multiple groups.

Dynamic operations with respect to key-derivation graphs. Let $N \in \mathbb{N}$, $\mathcal{S} = \{S_1, \ldots, S_k\} \subseteq 2^{[N]}$, and let $\mathcal{G}$ be a key-derivation graph generated by our algorithm on input $(N, \mathcal{S})$. Further, let $\mathcal{G}_{\text{lat}} = (\mathcal{V}_{\text{lat}}, \mathcal{E}_{\text{lat}})$ be the corresponding lattice graph. Consider a party $n \in [N]$ with index set $I = I(n) = \{i \in [k] \mid n \in S_i\}$. As discussed in Section 5 n's node is a leaf of the Huffman tree rooted at $v_I \in \mathcal{V}_{\text{lat}}$. Our goal is to support operations $\mathsf{Add}(n, i)$ which refreshes group key sk_{S_i} and gives n access to it, and $\mathsf{Rem}(n, i)$ which removes n from group i, i.e., replaces sk_{S_i} with a key not known to n.

Conceptually, $\mathsf{Add}(n, i)$ and $\mathsf{Rem}(n, i)$ correspond to changing n's index set I to $I' = I \cup \{i\}$ or $I' = I \setminus \{i\}$ respectively.[8] We can break down this process in two steps. The first corresponds to changing the structure of the key-derivation graph. The leaf v_n needs to be removed from the tree rooted at v_I while a new leaf v'_n (Owned by party n) has to be added to the tree rooted at $v_{I'}$. If $I' = \emptyset$ no leaf is added. It is possible that the node $v_{I'}$, i.e, a node that has paths to exactly the nodes v_{S_j} for $j \in I'$, is not yet part of the lattice graph and has to be added as well. Note that in the case $I' = I \cup \{i\}$ the new node $v_{I'}$ can be connected in the lattice graph using the two edges $(v_{I'}, v_I)$ and $(v_{I'}, v_{\{i\}})$. If $I' = I \setminus \{i\}$ more edges might be necessary. After $\mathcal{G}_{\text{lat}}$ has been updated the Huffman trees with leaf $v_{I'}$ have to be updated.

As, after changing the structure of $\mathcal{G}$, the invariant that every party only knows the secret keys corresponding to the descendants of their leaf no longer holds, in a second step key material needs to be refreshed. More precisely all keys corresponding to descendants $\mathcal{D}(v_n)$ of n's former leaf have to be replaced with fresh ones, and similarly all key material corresponding to $\mathcal{D}(v'_n)$ has to be refreshed starting with leaf key $\mathsf{sk}_{v'_n}$ that has to be accessible to n. We discuss how this can be implemented for CGKA and Multicast in greater detail below.

Continuous group-key agreement. In the setting of continuous group-key agreement there exists no central authority that holds all secret keys and administers structural changes in the groups. Accordingly, the action of adding party n to group S_i or removing n from S_i has to be initiated by a party m. To be able to do so without having party m sample key material for nodes outside of $\mathcal{D}(v_m)$, we will rely on the techniques of *blanking* paths and *unmerged leaves* of [4]. To every node v in the key-derivation graph $\mathcal{G}$ we associate a flag $\texttt{blanked} \in \{0, 1\}$ and a list of unmerged leaves $\text{Unm}(v)$. Finally the *resolution* $\text{Res}(v)$ of v is defined as follows.

$$\begin{aligned} \text{Res}(v) &= \{v\} && \text{if } \texttt{blanked}(v) = 0 \\ \text{Res}(v) &= \textstyle\bigcup_{v' \in \mathcal{P}(v)} \text{Res}(v') && \text{else} \end{aligned}$$

where $\mathcal{P}(v)$ denotes the parents of v in $\mathcal{G}$. Intuitively, $\text{Unm}(v)$ indicates leaves below v that have not yet been integrated in the key derivation graph, and $\text{Res}(v)$ is used to address all leaves under v with a minimal set of unblanked nodes.

As discussed above, adding or removing user n from a group proceeds in two steps, the first computing structural changes in $\mathcal{G}$, the second refreshing key material. Let v_n denote the "old" leaf of n and v'_n the new leaf to be added to the Huffman tree with root $v_{I'}$.

To carry out the first step, firstly, v_n is removed from $\mathcal{G}$ and all remaining nodes v in $\mathcal{D}(v_n) \setminus \{v_n\}$ blanked, i.e. $\texttt{blanked}(v) \leftarrow 1$, the keys $(\mathsf{pk}_v, \mathsf{sk}_v)$ deleted, and the resolution $\text{Res}(v)$ updated accordingly. Secondly, the new leaf v'_n is added to the Huffman tree rooted at $v_{I'}$, and for every $v \in \mathcal{D}(v'_n)$ the list of unmerged leaves is updated to $\text{Unm}(v) \leftarrow \text{Unm}(v) \cup \{v'_n\}$. Finally, if the operation was of the form $\mathsf{Add}(n, i)$ additionally the group key corresponding to i is deleted. Note that in the setting where the whole structure of $\mathcal{G}$ is known to the initiating party m, all changes can be computed by m and then communicated to the remaining parties via the delivery server. In a setting where users only know the part of $\mathcal{G}$ relevant to them, i.e., $\mathcal{D}(v_m)$ and the public keys of the co-parents with respect to $\mathcal{D}(v)$, m simply poses a request of the form $\mathsf{Add}(n, i)$ or $\mathsf{Rem}(n, i)$ to the server, which in turn computes the changes in $\mathcal{G}$ and sends personalized messages to all parties.

As to the second step, note that all group keys corresponding to $I \cup I'$ have been deleted. Thus, in order to resume communication in group $S_i \in I \cup I'$ a member m of this group has to perform an update - this is similar

[8] One could also imagine a more general operation $\mathsf{Change}(n, I')$ subsuming Add and Rem which changes n's index set to $I' \subseteq [k]$. The techniques of this section easily extend to this setting.

to the case of [4]. We now highlight how updates with respect to blanked nodes and unmerged leaves are computed compared to the version for a static key-derivation graph described in Section 3.2. Let $\mathcal{T}_m$ be the (in the case of our algorithm, unique) spanning tree of $\mathcal{D}(v)$. Then, starting from leaf v_m, new key pairs $(\mathsf{pk}_v, \mathsf{sk}_v)$ are generated from seeds Δ_v for all $v \in \mathcal{D}(v)$ in the way defined in Section 3.2. The set of ciphertexts corresponding to v is computed starting from leaf v_m as follows. For every co-parent $v' \in \mathcal{CP}(v, \mathcal{T}_m)$ with respect to the spanning tree and every $v'' \in \mathrm{Res}(v') \cup \mathrm{Unm}(v')$, a ciphertext $\mathsf{Enc}(\mathsf{pk}_{v''}, \Delta_v)$ is generated. After computing the ciphertexts corresponding to v, the flag $\texttt{blanked}(v)$ is set to 0, the resolution is recomputed accordingly, and the set of unmerged leaves is updated to $\mathrm{Unm}(v) \leftarrow \emptyset$. The Update message consists of all ciphertexts.

Consider the case that an operation $\mathsf{Rem}(n, i)$ was carried out followed by an update by party $m \in S_i$. Since by correctness $v_{S_i} \in \mathcal{D}(v_m)$, the group key for S_i was refreshed. Further, all nodes with key material known to n were blanked (except for the new leaf v'_n, which does not have a path to v_{S_i}). This implies that all new keys generated by m were encrypted to keys not known by n, and so, n does not have access to the new group key for S_i. Note that all users in $S_i \setminus \{n\}$ have a path from their leaf to one of the v'' and hence are able to derive the new group key.

Now assume that a user $m \in S_j$ where j is an element of n's new index set I', performed an update. Then there must exist a node in $\mathcal{D}(v_m) \cap \mathcal{D}(v'_n)$. All elements of $\mathcal{D}(v'_n)$ contain v'_n as an unmerged leaf. Thus m must have encrypted a seed to $\mathsf{pk}_{v'_n}$ from which n can derive the new group key of S_j. A similar argument shows that the algorithm works as intended also for operation $\mathsf{Add}(n, i)$.

Summing up, if an add or remove operation for party n was carried out and I' and v'_n denote $n's$ new index set and leaf respectively, then:

(i) If a party $m \in S_j$ performs an update, then n can derive the group key of S_j exactly if $j \in I'$.

(ii) The graph invariant still holds, i.e., if n knows the secret key corresponding to node $v \in \mathcal{G}$ then it must hold that $v \in \mathcal{D}(v'_n)$.

Multicast. In the setting of multicast encryption, a central authority holds all keys k_v with $v \in \mathcal{G}$ and administers changes in the group structure. This makes adding users to or removing them from groups considerably easier. As discussed above, assume that the index set of party n changes from I to I' and let v_n and v'_n denote the old deleted leaf and the new leaf, respectively. To refresh the keys in $\mathcal{D}(v_n)$ the central authority derives them from a random seed and computes the corresponding ciphertexts as discussed in Section 3.2. Similarly, starting from a seed $\Delta_{v'_n}$ all key material for nodes in $\mathcal{D}(v'_n)$ is resampled and corresponding ciphertexts are prepared. Note that n needs to be given access to $\Delta_{v'_n}$. An easy way to is to simply update the old leaf seed by hashing it with a secure hash function, i.e, by setting $\Delta_{v'_n} \leftarrow \mathsf{H}'(\Delta_{v_n})$. Now n can compute the new seed locally.

7 Lower Bound on the Update Cost of CGKA

In this Section we prove a lower bound on the average update cost of continuous group key agreement schemes for multiple groups. As an intermediate step we will further prove a bound on the update cost of key-derivation graphs. To this aim, we follow the approach of Micciancio and Panjwani [14], who analyzed the worst-case communication complexity of multicast key distribution in a *symbolic* security model, where cryptographic primitives are considered as abstract data types. We will first recall their security model, adapt it to CGKA, and then prove how to extend their results to our setting. In Appendix B using a similar approach we prove a lower bound for multicast encryption.

7.1 Symbolic Model

We first define a symbolic model in the style of Dolev and Yao [10] for CGKA schemes. It follows the approach of Micciancio and Panjwani [14], but as it admits the uses of public-key encryption also includes elements of the model of Bienstock et al. [6] who analyze the communication cost of concurrent updates in CGKA schemes.

Building blocks. We restrict the analysis to schemes that are constructed from the following three primitives. Note that our construction is a special case of the constructions analyzed in this section.

- *Public-key Encryption:* Let $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ denote a public-key encryption scheme, where
  - KGen on input of secret key sk returns the corresponding public key pk.
  - Enc takes as input a public key pk and a message m, and outputs a ciphertext $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$.
  - Dec takes as input a secret key sk and a ciphertext c, and outputs a message $\mathsf{m} = \mathsf{Dec}(\mathsf{sk}, c)$. We assume perfect correctness: $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{m})) = \mathsf{m}$ for all sk, $\mathsf{pk} = \mathsf{KGen}(\mathsf{sk})$, and messages m.
- *Pseudorandom generator:* The algorithm G takes as input a secret key sk and expands it to a sequence of keys $\mathsf{G}_0(\mathsf{sk}), \ldots, \mathsf{G}_\ell(\mathsf{sk})$.
- *Secret sharing:* Let S, R denote the sharing and recovering procedures of a secret sharing scheme: For some access structure $\Gamma \subseteq 2^{[h]}$, the algorithm S takes as input a message m and outputs a set of shares $\mathsf{S}_1(\mathsf{m}), \ldots, \mathsf{S}_h(\mathsf{m})$ such that for any $I \in \Gamma$ it holds $\mathsf{R}(I, \{S_i(\mathsf{m})\}_{i \in I}) = \mathsf{m}$, but for any $I \not\subseteq \Gamma$ the message m cannot be recovered from $\{S_i(\mathsf{m})\}_{i \in I}$.

We consider the following data types that can be derived from other objects according to the following rules.

Data type		Grammar rules
Message m	$\leftarrow$	$\mathsf{sk}, \mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{m}), S_1(\mathsf{m}), \ldots, S_h(\mathsf{m})$
Public key pk	$\leftarrow$	$\mathsf{KGen}(\mathsf{sk})$
Secret key sk	$\leftarrow$	$R, \mathsf{G}_0(\mathsf{sk}), \ldots, \mathsf{G}_\ell(\mathsf{sk})$

To describe the information that can be recovered from a set of messages M, the *entailment relation* is defined by the following rules:

$$
\begin{array}{rcl}
\mathsf{m} \in M & \Rightarrow & M \vdash \mathsf{m} \\
M \vdash \mathsf{sk} & \Rightarrow & M \vdash \mathsf{G}_0(\mathsf{sk}), \ldots, \mathsf{G}_l(\mathsf{sk}) \\
M \vdash \mathsf{Enc}(\mathsf{pk}, \mathsf{m}), \mathsf{sk} : \mathsf{pk} = \mathsf{KGen}(\mathsf{sk}) & \Rightarrow & M \vdash \mathsf{m} \\
\exists I \in \Gamma : \forall i \in I : M \vdash \mathsf{S}_i(\mathsf{m}) & \Rightarrow & M \vdash \mathsf{m}
\end{array}
$$

By restricting to these relations we essentially assume *secure* encryption and secret sharing schemes. Examples and further comments (in the setting of multicast encryption) can be found in [14, Section 3.2]. The set of messages which can be recovered from M using relation $\vdash$ is denoted by $\mathsf{Rec}(M)$.

Continuous group-key agreement. We now define continuous group-key agreement protocols in the symbolic model. We consider the case of CGKA for a static system of users $[N]$ and groups $S_1, \ldots, S_k \subseteq [N]$. Note that a lower bound for schemes in this setting in particular also excludes schemes which allow dynamic operations, i.e., adding and removing users from groups.

A CGKA scheme for $[N]$ and $S_1, \ldots, S_k$ specifies two procedures:

- Initially, Setup assigns each user $n \in [N]$ a personal set SK_n^0 of secret keys. Furthermore, Setup generates a set $\mathrm{msgs}(0)$ of so-called *rekey* messages to establish for every group S_j a group secret key $\mathsf{sk}_{S_j}^0$. We require that the initial sets of personal keys consist of uniformly random keys, and that for all $n' \neq n$ and $\mathsf{sk} \in \mathsf{SK}_n^0$ we have $\mathsf{sk} \notin \mathsf{Rec}(\mathsf{SK}_{n'}^0, \mathrm{msgs}(0))$.

- In round t, the algorithm Update takes as input a user identity $n \in [N]$, establishes new sets $\mathsf{SK}^t_{n'}$ for all users n', and outputs some rekey messages $\mathrm{msgs}(t)$ to establish for every group S_j an epoch t group key $\mathsf{sk}^t_{S_j}$. We do *not* require the new sets and group keys to be distinct from the ones of round $t-1$. We denote the set of new uniformly random keys that were generated during the update procedure by the updating party by F^t_n.

Note that the only party generating new keys during update t is the updating party n. For ease of notation we define $\mathsf{F}^t_{n'} = \emptyset$ for all $n' \neq n$, and set $\mathsf{F}^0_{n'} = \emptyset$ for all n'.

For *correctness*, we require that, (a) at all times members of a group are able to derive the current group key from their set of personal keys and the sent messages, and (b) that if some user updated in round t then all users are able to derive their new set of personal keys from their old one, the sent messages, and in the case of the updating party the new keys generated during the update. The latter condition accounts for the fact that changes to a user's set of personal keys need to be communicated to them.

More precisely, for (a) we require that for any subgroup structure and any sequence of updating users $(n_1, \ldots, n_t)$, for all $j \in [k]$ each member n of subgroup S_j can recover $\mathsf{sk}^t_{S_j}$:

$$\mathsf{sk}^t_{S_j} \in \mathsf{Rec}\Big(\mathsf{SK}^t_n \cup \bigcup_{\iota \in [t]_0} \mathrm{msgs}(\iota)\Big) .$$

For (b) we require that for any subgroup structure and any sequence of updating users $(n_1, \ldots, n_t)$, we have for all n that

$$\mathsf{SK}^t_n \subseteq \mathsf{Rec}\Big(\mathsf{SK}^{t-1}_n \cup \mathsf{F}^t_n \cup \bigcup_{\iota \in [t]_0} \mathrm{msgs}(\iota)\Big) .$$

For security, we assume the minimal requirement of *post-compromise security* (PCS), which essentially says that users can recover from compromise, which leaks their state and the keys generated during the time period of being compromised, by updating. Note that a lower bound in this setting in particular excludes protocols achieving stronger security notions desired in practice like post compromise forward security [3].

More precisely, we formalize PCS as the condition that no group key can be recovered from members outside the group and/or members' personal keys and the keys generated by them before their last update. To this end for round t and user $n \in [N]$ let $t_{\mathrm{up}}(t, n)$ denote the round in which n performed their last update, where we set $t_{\mathrm{up}}(t, n) = 0$ if no such update occurred. I.e., we require that for any group system, any update pattern, in every round t we have that

$$\mathsf{sk}^t_{S_j} \notin \mathsf{Rec}\Big(\bigcup_{\substack{n \in [N] \setminus S_j, \\ t' \in [t]_0}} (\mathsf{SK}^{t'}_n \cup \mathsf{F}^{t'}_n) \cup \bigcup_{n \in S_j} \bigcup_{t' \in [t_{\mathrm{up}}(t,n)-1]_0} (\mathsf{SK}^{t'}_n \cup \mathsf{F}^{t'}_n) \cup \bigcup_{t' \in [t]_0} \mathrm{msgs}(t')\Big) .$$

Note that in the definition above excluding all sets of personal secret keys since a user's last update is necessary even in the case that another user's update might have replaced them before round t, as otherwise SK^t_n and in turn $\mathsf{sk}^t_{S_j}$ could trivially be recovered by the two correctness conditions.

Our goal is to derive a lower bound on the communication complexity of CGKA schemes achieving PCS, i.e., the number of messages $|\cup_{t' \in [t]_0} \mathrm{msgs}(t')|$ sent by the protocol.

Key Graphs. The execution of any CGKA scheme can be reflected by a graph structure representing recoverability of the keys involved (cf. [14]). To define this graph, we first need to recall the definition of *useful* keys and messages.

A secret key sk is called *useless at time t* if it can be recovered from old key material, i.e., if

$$\mathsf{sk} \in \mathsf{Rec}\Big(\bigcup_{n \in [N]} \bigcup_{t' \in [t_{\mathrm{up}}(t,n)-1]_0} (\mathsf{SK}^{t'}_n \cup \mathsf{F}^{t'}_n) \cup \bigcup_{t' \in [t]_0} \mathrm{msgs}(t')\Big) ,$$

otherwise sk is called *useful*. As we will show below, if a CGKA scheme satisfies correctness and post-compromise security, then for all $t \in \mathbb{N}$, $n \in [N]$, $j \in [k]$ it must hold that at least one of the user's personal keys $\mathsf{sk}^t_n \in \mathsf{SK}^t_n$ as well as all group keys $\mathsf{sk}^t_{S_j}$ are useful at time t.

To decide whether a message is useful, one has to consider the information it contains, where messages can be arbitrarily nested applications of encryption Enc and secret sharing S. Thus, a message m is said to *encapsulate* a (pseudo)random key sk if $\mathsf{m} = e_1(e_2(\dots(e_j(\mathsf{sk}))\dots))$ where $e_i = \mathsf{Enc}_{\mathsf{pk}_i}$ or $e_i = \mathsf{S}_{h_i}$ (for some public key pk_i and $h_i \in [h]$). A message is then called *useful* if it encapsulates a useful key.

Definition 2 (Key graph [14])**.** *The* key graph $\mathcal{KG}_t = (\mathcal{V}_t, \mathcal{E}_t)$ *for a CGKA scheme at time t is defined as follows. $\mathcal{V}_t$ consists of all the keys that are useful at time t, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ consists of all ordered pairs $(\mathsf{sk}_1, \mathsf{sk}_2)$ such that one of the following is true:*

1. *There exists $j \in [l]$ auch that $\mathsf{sk}_2 = \mathsf{G}_j(\mathsf{sk}_1)$.*
2. *There exists a message $\mathsf{m} \in \bigcup_{j \in [t]_0} \mathrm{msgs}(j)$ with $\mathsf{m} = e_1(\mathsf{Enc}(\mathsf{pk}_1, e_2(\mathsf{sk}_2)))$ with $\mathsf{pk}_1 = \mathsf{KGen}(\mathsf{sk}_1)$. Here e_1 and e_2 are some sequences of encryption and secret sharing, and we require that e_2 does not contain any encryption under a public key that has a matching secret key that is useful at time t.*

Edges of the second type are called communication edges*.*

One can show that for any node sk in $\mathcal{KG}$ there is at most one edge of the first type incident to sk (the proof is analogous to [14, Proposition 1]). Note that edges of the first type do not incur any communication cost, while edges of the second type require at least one message. Thus, in the following we will be interested in the number of communication edges. To this aim, we prove the following properties of key graphs. In particular, we show that even if a CGKA scheme does not rely on the use of a fixed key-derivation graph as discussed in Section 3, after every update the key graph must still have the properties of Definition 1.

We will rely on the following Lemma that can be proved analogously to [14, Lemma 1].

Lemma 7. *Consider a secure and correct CGKA scheme for $N \in \mathbb{N}$, $S_1, \dots, S_k \subseteq [N]$. Then, for any $t \in \mathbb{N}$ and sequence of updates $(n_1, \dots, n_t)$, the corresponding key graph $\mathcal{KG}_t$ satisfies the following. For every set of keys SK, and key sk_2 that is useful at time t, such that $\mathsf{sk}_2 \in \mathsf{Rec}\left(\mathsf{SK} \cup \bigcup_{t' \in [t]_0} \mathrm{msgs}(t')\right)$, there exists a useful $\mathsf{sk}_1 \in \mathsf{SK}$ such that there is a path from sk_1 to sk_2 in $\mathcal{KG}_t$ that only consists of keys sk with $\mathsf{sk} \in \mathsf{Rec}\left(\mathsf{SK} \cup \bigcup_{t' \in [t]_0} \mathrm{msgs}(t')\right)$.*

Note that the converse of Lemma 7 is not true, since, for example, a message $\mathsf{Enc}(\mathsf{pk}_1, \mathsf{S}_1(\mathsf{sk}_2))$ with useful keys $\mathsf{sk}_1, \mathsf{sk}_2$ and $\mathsf{pk}_1 = \mathsf{KGen}(\mathsf{sk}_1)$ incurs an edge $(\mathsf{sk}_1, \mathsf{sk}_2)$ while sk_2 can only be recovered from sk_1 if $\{1\} \in \Gamma$.

7.2 Lower Bound on the Average Update Cost.

The communication complexity of a CGKA scheme after t updates is given by $\left|\bigcup_{t' \in [t]_0} \mathrm{msgs}(t')\right|$. To measure the efficiency of the scheme we will consider the *amortized communication complexity*

$$\mathrm{Com}_{\mathrm{A}} := \Big|\bigcup_{t' \in [t]_0} \mathrm{msgs}(t')\Big| / t \ .$$

We now are ready to compute a bound on the expectation of $\mathrm{Com}_{\mathrm{A}}$ in the scenario where in every round the updating party is chosen uniformly at random. In Appendix B we prove an analogous bound for multicast encryption that improves on [14, Theorem 1] in two aspects. It generalizes the bound to the setting of several, potentially overlapping groups, and further gives a bound on the *average* communication complexity of updates opposed to a worst case bound.

Theorem 3. *Consider a CGKA scheme CGKA for $N \in \mathbb{N}$, $S_1, \dots, S_k \subseteq [N]$ that is secure in the symbolic model. Then the expected amortized average communication cost after t updates is bounded from below by*

$$\mathbb{E}[\mathrm{Com}_{\mathrm{A}}] \geq (1 - 1/t) \cdot \frac{1}{N} \sum_{\emptyset \neq I \subseteq [k]} |P_I| \cdot \log(|P_I|) \ .$$

and the asymptotic (in the number of update operations) update cost of the protocol is at least $\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$.

Proof. We prove the result by showing that the average communication complexity after the tth update has size at least $(t-1)\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$. To this end, we will show that with every update on average at least $\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$ useful messages become useless. We will rely on the following claim.

Claim 2. *There exists a CGKA scheme* CGKA' *that is secure in the symbolic model such that:*

1. *If* CGKA *and* CGKA' *are executed with respect to the same update pattern, then their communication costs coincide.*
2. *Consider a sequence of* t *updates. For every* $t'<t$ *there exist a subgraph* $\mathcal{H}'_{t'}$ *of the keygraph* $\mathcal{G}'_{t'}$ *of* CGKA' *at time* t'*, such that for every* $n\in[N]$ *there exists a set* $V_n^{t'}$ *of useful sources of* $\mathcal{H}'_{t'}$ *with* $V_n^{t'}\subseteq\bigcup_{t''\in[t']_0}(\mathsf{SK}_n^{t''}\cup\mathsf{F}_n^{t''})$ *such that* $(\mathcal{H}'_{t'},\{V_n^{t'}\})$ *satisfies the requirements of Lemma 3.*

Before proving the claim we show that it implies Theorem 3. To this end, recall, that at most one of the edges incident to a node in a key graph is not a communication edge. For $t'<t$ consider the key graph $\mathcal{G}'_{t'}$. By applying Lemma 3 to the subgraph $\mathcal{H}'_{t'}$ of $\mathcal{G}'_{t'}$ the number of useful messages encapsulating secret keys that can be reached from useful keys in $\bigcup_{t''\in[t']_0}(\mathsf{SK}_n^{t''}\cup\mathsf{F}_n^{t''})$ is on average at least $\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$.

Note that by definition of PCS all useful keys in $\bigcup_{t''\in[t']_0}(\mathsf{SK}_n^{t''}\cup\mathsf{F}_n^{t''})$ become useless if party n updates in the $(t'+1)$th round. By Lemma 7 all descendants of these keys and in turn messages encapsulating descendants become useless as well. We obtain that with update $(t'+1)$ on average at least $\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$ messages become useless. By linearity of expectation and since useless messages never become useful again this implies that after the t updates on average at least $(t-1)\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$ messages must have been sent in CGKA'. As CGKA by Claim 2 has the same communication cost as CGKA' this bound carries over to it. Now dividing by t yields the claim of the theorem.

All that remains to do is to prove Claim 2. We define CGKA' to be the scheme that works uses the same initialization procedure as CGKA and computes updates in the same way, except that whenever a uniformly random secret key sk is generated in CGKA then CGKA' samples a uniformly random key sk' and sets $\mathsf{sk}\leftarrow\mathsf{G}_0(\mathsf{sk}')$ instead of the uniformly random key; this modified key sk is then used just the same as in CGKA in all further operations.

Note that the communication cost of both schemes coincides since CGKA' only makes additional calls to the pseudorandom generator but no additional use of the encryption and secret sharing schemes and that CGKA' preserves correctness. Further, CGKA' is secure since CGKA is secure: To see this, note that in the symbolic model there is no difference between a uniformly random key and a pseudorandom key, as long as the seed of the latter is not revealed. But the additional seeds sk' which we introduce in CGKA' are never used in any messages, nor are they used to derive any further keys; they only occur in the sets F_n^t where they replace the keys sk. Thus, security of CGKA' indeed directly follows from security of CGKA.

We now show that the second part of Claim 2 holds as well. In fact, for a sequence of t updates we will prove the following stronger statement. For all $t'<t$ (a) there exists a subgraph $\mathcal{H}'_{t'}$ of $\mathcal{G}'_{t'}$ with distinct nodes v_{S_i} and pairwise distinct sets $V_n^{t'}\subseteq\bigcup_{t''\in[t']_0}(\mathsf{SK}_n^{t''}\cup\mathsf{F}_n^{t''})$ of sources such that

$$n\in S_i\quad\Rightarrow\quad\exists v_n\in V_n^{t'}\text{ such that there is a path from }v_n\text{ to }v_{S_i}\ ,$$

and that (b) for all $n\neq n'$ and $v\in V_n^{t'}$ it holds that $v\notin\mathsf{Rec}(V_{n'}^{t'}\cup\mathsf{SK}_{n'}^{t'-1})$.

We argue inductively in t' that a subgraph and sets with the properties (a) and (b) must always exist. First consider the case $t'=0$. Note that the group keys $v_{S_j}=\mathsf{sk}_{S_j}^0$ by definition are useful at time 0. Fix S_j and let $n\in S_j$. By correctness and Lemma 7 there exists a useful $v_{n,j}\in\mathsf{SK}_n^0$ that has a path to v_{S_j} in $\mathcal{G}'_0$. We define $V_n^0=\{v_{n,j}\mid j:n\in S_j\}$ and $\mathcal{H}'^0$ to be the subgraph of $\mathcal{G}'^0$ induced by the union over n and j of paths from $v_{n,j}$ to v_{S_j}. Then the correctness condition of (a) holds and we only have to show that the V_n^0 consist of pairwise distinct sources. By definition we have $v_{n,j} \notin \mathsf{Rec}(\mathsf{SK}_{n'}^0)$ for $n' \neq n$ implying that the V_n^0 are pairwise distinct, and further by Lemma 7 that $v_{n,j}$ in $\mathcal{H}_0'$ cannot be reached by any $v_{n',j'}$ with $n' \neq n$. Note that if $v_{n,j}$ can be reached by $v_{n,j'}$ with $j' \neq j$ then we can simply remove $v_{n,j}$ from V_n^0 without changing the correctness condition. Thus, we end up with pairwise disjoint sets V_n^0 of sources and (a) holds.

Now assume that (a) and (b) hold for all $t'' < t'$. Let $n_{t'}$ denote the party that issued update t'. First consider a party $n \neq n_t$ and a group S_j with $n \in S_j$. Note that by correctness and security the group key $v_{S_j} = \mathsf{sk}_{S_j}^t$ is useful at time t'. Further, by correctness we have

$$\mathsf{sk}_{S_j}^{t'} \in \mathsf{Rec}\Big(\mathsf{SK}_n^{t'} \cup \bigcup_{t'' \in [t']_0} \mathrm{msgs}(t'')\Big) \quad \text{and} \quad \mathsf{SK}_n^{t'} \subseteq \mathsf{Rec}\Big((\mathsf{SK}_n^{t'-1} \cup \mathsf{F}_n^{t'}) \bigcup_{t'' \in [t']_0} \mathrm{msgs}(t'')\Big) \ .$$

Thus, by Lemma 7 there exists a useful node $v \in \mathsf{SK}_n^{t'}$ with a path to v_{S_j} and a useful node $v' \in \mathsf{SK}_n^{t'-1}$ that has a path to v, where we used that $\mathsf{F}_n^{t'} = \emptyset$. As v' already existed at time $t'-1$ and lies in $\bigcup_{t'' \in [t'-1]_0} (\mathsf{SK}_n^{t''} \cup \mathsf{F}_n^{t''})$ by the induction hypothesis there must exist a node $v_{n,S_j} \in V_n^{t'-1}$ that is a source in $\mathcal{H}_{t'-1}' \subseteq \mathcal{G}_{t'-1}'$ and has a path to v' and in turn to v and v_{S_j}. We set $V_n^{t'} = \{v_{n,S_j} \mid j : n \in S_j\}$.

Now consider the party $n = n_{t'}$ that issued update t' and let S_j be such that $n \in S_j$. By the first correctness property we have $\mathsf{sk}_{S_j}^{t'} \in \mathsf{Rec}(\mathsf{SK}_n^{t'} \cup \bigcup_{t' \in [t']_0} \mathrm{msgs}(t'))$. Since the node $v_{S_j} = \mathsf{sk}_{S_j}^{t'}$ is useful at time t' by Lemma 7 there exists a useful node $v \in \mathsf{SK}_n^{t'}$ with a path to v_{S_j}. Further, by the definition of PCS it is not possible that v can be recovered from $\mathsf{SK}_n^{t''}$ for any $t'' < t'$ and thus, must have been generated during the t'th update. More precisely, since n is the updating party, by security all elements of $\mathsf{SK}_n^{t'-1}$ are useless at time t' and since by correctness $v \in \mathsf{Rec}(\mathsf{SK}_n^{t'-1} \cup \mathsf{F}_n^{t'})$ we obtain by Lemma 7 that there exists useful $v_{n,j} \in \mathsf{F}_n^{t'}$ that has a path to v and in turn to v_{S_j}. Note that $v_{n,j}$ by definition of $\mathsf{F}_n^{t'}$ is a uniformly random secret key. By construction of CGKA' the only operation applied to $v_{n,j}$ was an application of G_0, which in particular implies that it never was encrypted under any key. Thus $v_{n,j}$ is a source in $\mathcal{G}_{t'}'$ and we can define $V_n^{t'} = \{v_{n,j} \mid j : n \in S_j\}$.

Now we can define $\mathcal{H}_{t'}'$ to be the subgraph of $\mathcal{G}_{t'}'$ induced by the union over n and j of paths from $v_{n,j}$ to v_{S_j}. Note that for any party $n \neq n_{t'}$ that did not update in round t' any $v_{n,j} \in V_n^{t'}$ can only be reachable from some other node $v \in (V_{n'}^{t'})$ with $n \neq n'$ in $\mathcal{H}_{t'}$ if during the t'th update it was encrypted under some key that can be recovered from $V_{n'}^{t'-1} \subseteq V_{n'}^{t'-1} \cup \mathsf{SK}_{n'}^{t'-1}$. This however, would contradict induction hypothesis (b). Thus all elements of $V_n^{t'}$ must indeed be sources in $\mathcal{H}_{t'}$.

Finally, note that (b) holds as well: For $n \neq n_t$ this follows from the induction hypothesis and correctness and for n_t as discussed above by construction of CGKA'.

□

8 Open problems

We conclude by discussing some open problems.

8.1 Optimal Key-derivation Graphs

Unfortunately we are not able to tell how far from optimal the solutions generated by Algorithm 1 are for concrete group systems. We consider it an interesting open question to resolve this issue.

General kdgs. We first discuss this problem in its general form. I.e., given a system $\mathcal{S} = \{S_1, \ldots, S_k\}$ of subgroups of the set $[N]$ of users compute the key-derivation graph for $\mathcal{S}$ (as defined in Definition 1) that has minimal update cost. The question whether a polynomial time algorithm for solving this problem exists can be naturally asked in various ways. E.g., when polynomial means polynomial in the number of users N

(think of N being given in unary), or polynomial in a reasonable description of the set system $\mathcal{S}$, say, when we are given the sizes of all non-empty intersections of sets in $\mathcal{S}$. Here N can be exponential in the input length, so a potential solution would need to have a very succinct description. Algorithm 1 (for which we don't know whether it is optimal) can be turned into of the latter kind by using an implicit representation during the Huffman coding step.

We are thankful to one reviewer of this work, who pointed out an interesting connection of key-derivation graphs for a group system $\mathcal{S} = \{S_1, \ldots, S_k\}$ to the *disjunctive complexity* of $\mathcal{S}$, which, given variables $x_1, \ldots, x_N \in \{0, 1\}$, corresponds to the size of the smallest circuit of fanin-2 OR-gates computing

$$\bigvee_{i \in S_1} x_i, \ldots, \bigvee_{i \in S_k} x_i \ . \tag{12}$$

Note that circuits computing (12) correspond exactly to key-derivation graphs for $\mathcal{S}$. So the two problems differ only by the used metric; while disjunctive complexity counts the number of non-sources in the graph, the update cost of a kdg weighs each of these nodes by the number of sources below it. As there exist upper and lower bound on the disjunctive complexity of group systems (see e.g. [12]), we consider it an interesting open questions whether these can be used to establish bounds on the update cost of kdgs. We want to point out, however, that this metric might be not fine-grained enough to capture certain properties of kdgs: E.g., for $N \in \mathbb{N}$ the systems $\mathcal{S}_1 = \{[N]\}$ and $\mathcal{S}_2 = \{[1], [2], \ldots, [N]\}$ both have disjunctive complexity $N - 1$, but their total update costs as kdgs are of order $N \cdot \log(N)$ and N^2 respectively.

Lattice based kdgs. If we restrict our view to algorithms using to Boolean-lattice based graphs as defined in Section 5.4 and are willing to make simplifying assumptions the question of optimality translates to an optimization problem on graphs: We are (a) going to consider only lattice graphs $\mathcal{G}_{\text{lat}}$ where all nodes v are connected with their descendants $v' \in \mathcal{D}(v)$ by an *unique* path, and (b) in our analysis of the update cost assume that the algorithms second step (i.e., the generation of Huffman trees) is instead implemented with an idealized code that has average codeword length matching the entropy of the leaf distribution. This essentially corresponds to ignoring the terms of $+1$ in Lemma 1.

Recall that for groups system $\{S_1, \ldots, S_k\}$ the nodes $v_I \in \mathcal{V}_{\text{lat}}$ of a lattice graph correspond to index sets $I \subseteq [k]$ It is easy to see that correctness of $\mathcal{G}_{\text{lat}}$ together with condition (a) is equivalent to requiring that the only sinks in the graph are the singleton sets $\{i\}$ and that for every $v_I \in \mathcal{V}_{\text{lat}}$

$$I = I_1 \uplus \cdots \uplus I_\ell \tag{13}$$

holds, where $v_{I_1}, \ldots, v_{I_\ell}$ are the children of v_I and disjointness enforces unique paths.

The total update cost of a graph satisfying this property can be computed as follows. To every node v_I we associate the weight $w_I = \left|\bigcap_{i \in I} S_i \setminus \bigcup_{j \in [k] \setminus I} S_j\right|$ corresponding to the number of users exactly in the groups specified by I. Further, we inductively define the total weight t_I of v_I as

$$t_I = \begin{cases} w_I & \text{if } v_I \text{ is source} \\ w_I + \sum_{I' \colon v_{I'} \in \mathcal{P}(v_I)} t_{I'} & \text{else} \end{cases} ,$$

where $\mathcal{P}(v_I)$ denotes the set of parents of v_I. By assumptions (a) and (b), and Lemma 5 the update cost contributed by node v_I thus corresponds to

$$\text{Upd}(v_I) = t_I \log(t_I) - \sum_{I' \colon v_{I'} \in \mathcal{P}(v_I)} t_{I'} \log(t_{I'}) \tag{14}$$

and we end up with the following optimization problem on lattice graphs.

Problem 1. *Let $k \in \mathbb{N}$. Given weights $\{w_I\}_{I \subseteq [k]}$ with $w_I \in \mathbb{N}$ among the subgraphs of the Boolean lattice with respect to the power set of $[k]$ that satisfy Condition 13 find the subgraph $\mathcal{G}_{\text{lat}}$ of minimal total update cost*

$$\text{Upd}(\mathcal{G}_{\text{lat}}) = \sum_{I \subseteq [k]} \text{Upd}(v_I) \ .$$

We consider it an interesting open question whether Algorithm 1 solves this problem and, if not, to find an efficient algorithm that does.

8.2 Security

In this work we focused on the communication complexity of key-derivation graphs and only gave an intuition on their security. Security proofs for secure group messaging are typically quite complex, and protocols rely on additional mechanisms (e.g. confirmation tag, transcript hash, and parent hash) ensuring that users of the system can not be tricked into inconsistent views of the graph. We consider it an important open question, to adapt theses mechanisms to kdgs for several groups and give a formal security proof for the resulting CGKA protocols.

8.3 Efficiency of Dynamic Operations

As discussed in Section 6 the techniques of blanking and unmerged leaves can be adapted to key-derivation graphs in order to allow dynamic changes to the group membership. As is the case for singular groups, blanking and unmerged leaves decrease the efficiency of updates of a user n, since they destroy the binary structure of the graph resulting in potentially more than a single ciphertext per node in $\mathcal{D}(v_n)$ having to be generated. However, the graph gradually recovers from this, assuming that parties with update trees overlapping $\mathcal{D}(v_n)$ update. It is an interesting open question how the decrease in efficiency compares to that of the trivial algorithm.

References

[1] Y. Abu-Mostafa and R. McEliece. Maximal codeword lengths in huffman codes. *Computers & Mathematics with Applications*, 39(11):129 – 134, 2000.

[2] J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, Aug. 2020.

[3] J. Alwen, S. Coretti, D. Jost, and M. Mularczyk. Continuous group key agreement with active security. In R. Pass and K. Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 261–290. Springer, Heidelberg, Nov. 2020.

[4] R. Barnes, B. Beurdouche, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-11, Internet Engineering Task Force, Dec. 2020. Work in Progress.

[5] K. Bhargavan, R. Barnes, and E. Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups. May 2018.

[6] A. Bienstock, Y. Dodis, and P. Rösler. On the price of concurrency in group ratcheting protocols. In R. Pass and K. Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 198–228. Springer, Heidelberg, Nov. 2020.

[7] R. Canetti, J. A. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *IEEE INFOCOM'99*, pages 708–716, New York, NY, USA, Mar. 21–25, 1999.

[8] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 1802–1819. ACM Press, Oct. 2018.

[9] C. Cremers, B. Hale, and K. Kohbrok. Efficient post-compromise security beyond one group. Cryptology ePrint Archive, Report 2019/477, 2019. https://eprint.iacr.org/2019/477.

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

[11] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[12] S. Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012.

[13] T. T. Mapoka, S. Shepherd, R. Abd-Alhameed, and K. O. Anoh. Novel rekeying approach for secure multiple multicast groups over wireless mobile networks. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 839–844. IEEE, 2014.

[14] D. Micciancio and S. Panjwani. Optimal communication complexity of generic multicast key distribution. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 153–170. Springer, Heidelberg, May 2004.

[15] D. M. Wallner, E. J. Harder, and R. C. Agee. Key management for multicast: Issues and architectures. Internet Draft, Sept. 1998. http://www.ietf.org/ID.html.

[16] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, Feb. 2000.

[17] H. Zhong, W. Luo, and J. Cui. Multiple multicast group key management for the internet of people. *Concurrency and Computation: Practice and Experience*, 29(3):e3817, 2017. e3817 CPE-15-0502.R1.

A Direct Comparison of Trivial Algorithm and Algorithm 1

Tighter Analysis of Example 1. We now give an analysis of Example 1 that shows that at least in such a situation our algorithm performs always better than the trivial solution, even including rounding. For simplicity, assume n_1 is a power of 2 (but k is arbitrary). Consider the following hypothetical construction of $\mathcal{T}_1$ and $\mathcal{T}_{1,2}$: first build the complete binary tree and keep all k users in $S_{12} = S_1 \cap S_2$ to the right. Let v be the node highest up in the tree such that all its parents are in S_{12}. Now remove all users in S_{12} from the tree and call the result $\mathcal{T}_1$. Build a new binary tree $\mathcal{T}_{1,2}$ (as balanced as possible) from the nodes in S_{12} and attach it to node v. Clearly, all users in $S_1 \setminus S_2$ have path length $\log n_1$ in $\mathcal{T}_1$ and node v has path length $\log n_1 - \lfloor \log k \rfloor$. Also, all users in S_{12} have path length $< \lceil \log k \rceil < \lfloor \log k \rfloor + 1$ in $\mathcal{T}_{1,2}$. So we get

$$\mathbb{E}[\mathrm{len}(U_{\mathcal{T}_1})] = \frac{n_1 - k}{n_1} \log n_1 + \frac{k}{n_1}(\log n_1 - \lfloor \log k \rfloor) = \log n_1 - \frac{k}{n_1} \lfloor \log k \rfloor$$

and $\mathbb{E}[\mathrm{len}(U_{\mathcal{T}_{1,2}})] \leq \lfloor \log k \rfloor + 1$. Note that the same construction also works for $\mathcal{T}_2$ and yields

$$\mathbb{E}[\mathrm{len}(U_{\mathcal{T}_2})] = \log n_2 - \frac{k}{n_2} \lfloor \log k \rfloor$$

Since Huffman is optimal, creating $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_{1,2}$ by using Huffman cannot yield worse expected path lengths. Putting these together

$$\begin{aligned}\mathrm{Upd}(\mathcal{G}_{\mathrm{a1}}) &\leq n_1 \mathbb{E}[\mathrm{len}(U_{\mathcal{T}_1})] + n_2 \mathbb{E}[\mathrm{len}(U_{\mathcal{T}_2})] + k \mathbb{E}[\mathrm{len}(U_{\mathcal{T}_{1,2}})] \\ &\leq n_1 \log n_1 + n_2 \log n_2 - k(\lfloor \log k \rfloor - 1).\end{aligned}$$

Clearly, for $k \geq 2$ this is always negative.

Now we obtain an upper-bound in which the approach of the example and the use of Lemma 5 are combined in order to obtain a sufficient condition under which $\mathcal{G}_{\mathrm{a1}}$ outperforms $\mathcal{G}_{\mathrm{triv}}$. We generalize the example to build the trees that correspond to nodes of the form $v_{\{i\}}$ in $\mathcal{V}_{\mathrm{lat}}$ and then use Lemma 5 for the rest of nodes in $\mathcal{V}_{\mathrm{lat}}$.

Comparison of Trivial Algorithm and Algorithm 1. Let $S_1, \ldots, S_s \subseteq [N]$ and $\mathcal{G}_{\mathrm{lat}} = (\mathcal{V}_{\mathrm{lat}}, \mathcal{E}_{\mathrm{lat}})$ be the corresponding lattice graph. Let $T_{\{i\}} := \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| - \left|\mathsf{S}(v_{\{i\}})\right| = \sum_{\substack{J \subseteq 2^{[s]} \text{ st} \\ v_{\{i\} \cup J} \in \mathcal{P}(v_{\{i\}}) \cap \mathcal{P}(v_J)}} \left|\mathsf{S}(\mathcal{A}(v_{\{i\} \cup J}))\right|$. For each $i \in [s]$ first build a binary tree (as balanced as possible) using all users in $\mathsf{S}(\mathcal{A}(v_{\{i\}}))$ and keeping $2^{\lfloor \log T_{\{i\}} \rfloor}$ users in $\bigcup_{v' \in \mathcal{P}(v_{\{i\}})} \mathsf{S}(\mathcal{A}(v'))$ to the right. Just as in the example, let v be the node highest up in the tree such that $\mathcal{P}(v) \subseteq \bigcup_{v' \in \mathcal{P}(v_{\{i\}})} \mathsf{S}(\mathcal{A}(v'))$. Then remove all users in $\bigcup_{v' \in \mathcal{P}(v_{\{i\}})} \mathsf{S}(\mathcal{A}(v'))$ from the tree and call the resulting tree $\mathcal{T}_{\{i\}}$. We build a Huffman tree, $\mathcal{T}^{aux}_{\{i\}}$, with $\left|\mathcal{P}(v_{\{i\}})\right|$ many leaves and weights $\mathsf{S}(\mathcal{A}(v'))$ for each $v' \in \left|\mathcal{P}(v_{\{i\}})\right|$ and attach it to v. Each leaf of $\mathcal{T}^{aux}_{\{i\}}$ corresponds to a node $v_{\{i\} \cup J}$ for some $J \subseteq 2^{[s]}$. We add an edge between each leaf of $\mathcal{T}^{aux}_{\{i\}}$ and the root of the corresponding $\mathcal{T}_{\{i\} \cup J}$. For $|I| \geq 2$ we just consider a Huffman tree $\mathcal{T}_I$. We can bound the update cost of $\mathcal{G}$ as $\mathrm{Upd}(\mathcal{G}) \leq \sum_{i \in [s]} (\mathrm{Upd}(\mathcal{T}_{\{i\}}) + \mathrm{Upd}(\mathcal{T}^{aux}_{\{i\}})) + \sum_{v_I \in \mathcal{V}_{\mathrm{lat}} \colon |I| \geq 2} \mathrm{Upd}(\mathcal{T}_I)$. We can upper-bound the update cost of $\mathcal{T}^{aux}_I$ using Lemma 5;

$$\mathrm{Upd}(\mathcal{T}^{aux}_{\{i\}}) \leq T_{\{i\}} \log T_{\{i\}} + T_{\{i\}} - \sum_{\substack{J \subseteq 2^{[s]} \text{ st} \\ v_{\{i\} \cup J} \in \mathcal{P}(v_{\{i\}}) \cap \mathcal{P}(v_J)}} \left|\mathsf{S}(\mathcal{A}(v_{\{i\} \cup J}))\right| \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\} \cup J}))\right|.$$

There exist $0 \leq a_{\{i\}}, b_{\{i\}} \leq 1$ such that $a_{\{i\}} + b_{\{i\}} = 1$, there are $a_{\{i\}} \left|\mathsf{S}(v_{\{i\}})\right|$ users that have path length at most $\lfloor \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rfloor$ in $\mathcal{T}_{\{i\}}$ and there are $b_{\{i\}} \left|\mathsf{S}(v_{\{i\}})\right|$ users that have path length $\lceil \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rceil$

in $\mathcal{T}_{\{i\}}$. The node v has path length at most $\lceil \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rceil - \lfloor \log T_{\{i\}} \rfloor$ in $\mathcal{T}_{\{i\}}$. Therefore we have

$$\begin{aligned}
\mathrm{Upd}(\mathcal{T}_{\{i\}}) \leq \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \Bigg(& \frac{a_I \left|\mathsf{S}(v_{\{i\}})\right|}{\left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right|} \lfloor \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rfloor + \frac{b_{\{i\}} \left|\mathsf{S}(v_{\{i\}})\right|}{\left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right|} \lceil \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rceil \\
& + \frac{T_{\{i\}}}{\left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right|} (\lceil \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rceil - \lfloor \log T_{\{i\}} \rfloor) \Bigg) \\
\leq \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \Bigg(& \frac{a_I \left|\mathsf{S}(v_{\{i\}})\right|}{\left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right|} \lfloor \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rfloor + \frac{b_{\{i\}} \left|\mathsf{S}(v_{\{i\}})\right|}{\left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right|} \lceil \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rceil \\
& + \frac{T_{\{i\}}}{\left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right|} (\lfloor \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rfloor + 1 - \lfloor \log T_{\{i\}} \rfloor) \Bigg) \\
\leq \mathrm{Upd}(& \mathcal{G}_{\mathrm{triv}} \text{ of } S_i) + T_{\{i\}}(1 - \lfloor \log T_{\{i\}} \rfloor)
\end{aligned}$$

The last inequality follows from the fact that there cannot be less than $b_{\{i\}} \left|\mathsf{S}(v_{\{i\}})\right|$ users whose path length is $\lceil \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| \rceil$ in the tree constructed by the trivial algorithm.

We can upper-bound the update cost of $\mathcal{T}_I$ for $|I| \geq 2$ using Lemma 5;

$$\mathrm{Upd}(\mathcal{T}_I) \leq |\mathsf{S}(\mathcal{A}(v_I))| (1 + \log |\mathsf{S}(\mathcal{A}(v_I))|) - \sum_{\substack{J \subseteq 2^{[s]} \text{ st} \\ v_{I \cup J} \in \mathcal{P}(v_I) \cap \mathcal{P}(v_J)}} |\mathsf{S}(\mathcal{A}(v_{I \cup J}))| \log |\mathsf{S}(\mathcal{A}(v_{I \cup J}))| .$$

We sum over all $I \subseteq 2^{[s]}$ such that $v_I \in V'$ and we get

$$\begin{aligned}
\mathrm{Upd}(\mathcal{G}_{\mathrm{a1}}) &\leq \sum_{i \in [s]} (\mathrm{Upd}(\mathcal{T}_{\{i\}}) + \mathrm{Upd}(\mathcal{T}^{aux}_{\{i\}})) + \sum_{v_I \in \mathcal{V}_{\mathrm{lat}} \colon |I| \geq 2} \mathrm{Upd}(\mathcal{T}_I) \\
&\leq \sum_{i \in [s]} \Bigg(\mathrm{Upd}(\mathcal{G}_{\mathrm{triv}} \text{ of } S_i) + 3T_{\{i\}} - \sum_{\substack{J \subseteq 2^{[s]} \text{ st} \\ v_{\{i\} \cup J} \in \mathcal{P}(v_{\{i\}}) \cap \mathcal{P}(v_J)}} \left|\mathsf{S}(\mathcal{A}(v_{\{i\} \cup J}))\right| \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\} \cup J}))\right| \Bigg) \\
&+ \sum_{v_I \in \mathcal{V}_{\mathrm{lat}} \colon |I| \geq 2} \Bigg(|\mathsf{S}(\mathcal{A}(v_I))| (1 + \log |\mathsf{S}(\mathcal{A}(v_I))|) - \sum_{\substack{J \subseteq 2^{[s]} \text{ st} \\ v_{I \cup J} \in \mathcal{P}(v_I) \cap \mathcal{P}(v_J)}} |\mathsf{S}(\mathcal{A}(v_{I \cup J}))| \log |\mathsf{S}(\mathcal{A}(v_{I \cup J}))| \Bigg)
\end{aligned}$$

Using the fact that $T_{\{i\}} \coloneqq \left|\mathsf{S}(\mathcal{A}(v_{\{i\}}))\right| - \left|\mathsf{S}(v_{\{i\}})\right| = \sum_{\substack{J \subseteq 2^{[s]} \text{ st} \\ v_{\{i\} \cup J} \in \mathcal{P}(v_{\{i\}}) \cap \mathcal{P}(v_J)}} \left|\mathsf{S}(\mathcal{A}(v_{\{i\} \cup J}))\right|$ yields

$$\begin{aligned}
\mathrm{Upd}(\mathcal{G}_{\mathrm{a1}}) &\leq \sum_{i \in [s]} \mathrm{Upd}(\mathcal{G}_{\mathrm{triv}} \text{ of } S_i) + \sum_{i \in [s]} \sum_{\substack{J \subseteq 2^{[s]} \text{ st} \\ v_{\{i\} \cup J} \in \mathcal{P}(v_{\{i\}}) \cap \mathcal{P}(v_J)}} \left|\mathsf{S}(\mathcal{A}(v_{\{i\} \cup J}))\right| (3 - \log \left|\mathsf{S}(\mathcal{A}(v_{\{i\} \cup J}))\right|) \\
&+ \sum_{v_I \in \mathcal{V}_{\mathrm{lat}} \colon |I| \geq 2} \Bigg(|\mathsf{S}(\mathcal{A}(v_I))| (1 + \log |\mathsf{S}(\mathcal{A}(v_I))|) - \sum_{\substack{J \subseteq 2^{[s]} \text{ st} \\ v_{I \cup J} \in \mathcal{P}(v_I) \cap \mathcal{P}(v_J)}} |\mathsf{S}(\mathcal{A}(v_{I \cup J}))| \log |\mathsf{S}(\mathcal{A}(v_{I \cup J}))| \Bigg)
\end{aligned}$$

For $|I| \geq 2$, the term $|\mathsf{S}(\mathcal{A}(v_I))| \log |\mathsf{S}(\mathcal{A}(v_I))|$ appears twice with a negative sign and once with a positive sign. Hence

$$\begin{aligned}
\mathrm{Upd}(\mathcal{G}_{\mathrm{a1}}) &\leq \mathrm{Upd}(\mathcal{G}_{\mathrm{triv}}) + \sum_{v_I \in \mathcal{V}_{\mathrm{lat}} \colon |I| = 2} |\mathsf{S}(\mathcal{A}(v_I))| (7 - \log |\mathsf{S}(\mathcal{A}(v_I))|) \\
&+ \sum_{\substack{v_I \in \mathcal{V}_{\mathrm{lat}} \colon |I| > 2 \\ \exists i \in [s] \colon v_I \in \mathcal{P}(v_{\{i\}})}} |\mathsf{S}(\mathcal{A}(v_I))| (4 - \log |\mathsf{S}(\mathcal{A}(v_I))|) + \sum_{\substack{v_I \in \mathcal{V}_{\mathrm{lat}} \colon |I| > 2 \\ \nexists i \in [s] \colon v_I \in \mathcal{P}(v_{\{i\}})}} |\mathsf{S}(\mathcal{A}(v_I))| (1 - \log |\mathsf{S}(\mathcal{A}(v_I))|)
\end{aligned}$$

In particular, if

- $|\mathsf{S}(\mathcal{A}(v'))| \geq 2$ for every $v_I \in \mathcal{V}_{\text{lat}}$ with $|I| > 2$ and such that $\nexists i \in [s]$ with $v_I \in \mathcal{P}(v_{\{i\}})$, and
- $|\mathsf{S}(\mathcal{A}(v'))| \geq 2^3 = 8$ for every $v_I \in \mathcal{V}_{\text{lat}}$ with $|I| > 2$ and such that $\exists i \in [s]$ with $v_I \in \mathcal{P}(v_{\{i\}})$, and
- $|\mathsf{S}(\mathcal{A}(v'))| \geq 2^7 = 128$ for every $v_I \in \mathcal{V}_{\text{lat}}$ with $|I| = 2$,

$\mathcal{G}_{\text{a1}}$ outperforms $\mathcal{G}_{\text{triv}}$. The first condition applies to nodes which do not correspond to the intersection of two of the original subsets. The second condition applies to nodes that correspond to the intersection of two subsets of $[N]$, of which exactly one is among the original subsets. The last condition applies to nodes with $|I| = 2$, that is, nodes that correspond to the intersection of S_i and S_j for some $i, j \in [s]$.

B Multicast Encryption Lower Bound

In this Section we prove a lower bound on the average update cost of multicast encryption schemes for multiple groups. To this aim, we follow the approach of Micciancio and Panjwani [14], who analyzed the worst-case communication complexity of multicast key distribution in a *symbolic* security model, where cryptographic primitives are considered as abstract data types. We will first recall their security model and then prove how to extend their results to our setting.

B.1 Symbolic Model

We restrict the analysis to schemes that are constructed from the following three primitives. Note that our construction is a special case of the constructions analysed in this section.

- *Encryption:* Let (E, D) denote a symmetric-key encryption scheme, where
  - E takes as input a secret key k and a message m, and outputs a ciphertext $c \leftarrow \mathsf{E}_\mathsf{k}(\mathsf{m})$,
  - D takes as input a secret key k and a ciphertext c, and outputs a message $\mathsf{m} = \mathsf{D}_\mathsf{k}(\mathrm{c})$. We assume perfect correctness, i.e. $\mathsf{D}_\mathsf{k}(\mathsf{E}_\mathsf{k}(\mathsf{m})) = \mathsf{m}$ for all keys k and messages m. Furthermore, the encryption scheme is secure, i.e., informally, without knowledge of the key k one cannot recover m from c.
- *Pseudorandom generator:* The algorithm G takes as input a key k and expands it to a sequence of keys $\mathsf{G}_0(\mathsf{k}), \ldots, \mathsf{G}_l(\mathsf{k})$, that are indistinguishable from a sequence $\mathsf{k}_0, \ldots, \mathsf{k}_l$ of uniformly random keys $\mathsf{k}_i \leftarrow R$ $(i \in [l]_0)$ without knowledge of the key k.
- *Secret sharing:* Let S, R denote the sharing and recovering procedures of a secret sharing scheme: For some access structure $\Gamma \subseteq 2^{[h]}$, the algorithm S takes as input a message m and outputs a set of shares $\mathsf{S}_1(\mathsf{m}), \ldots, \mathsf{S}_h(\mathsf{m})$ such that for any $I \in \Gamma$ it holds $\mathsf{R}(I, \{S_i(\mathsf{m})\}_{i \in I}) = \mathsf{m}$, but for any $I \not\subseteq \Gamma$ the message m cannot be recovered from $\{S_i(\mathsf{m})\}_{i \in I}$.

There are two types of data structures: messages and keys, which can be derived by repeatedly applying the above algorithms:

$$\mathsf{m} \leftarrow \{\mathsf{k},\, \mathsf{E}_\mathsf{k}(\mathsf{m}),\, \mathsf{S}_1(\mathsf{m}),\, \ldots,\, \mathsf{S}_h(\mathsf{m})\}, \quad \mathsf{k} \leftarrow \{R,\, \mathsf{G}_0(\mathsf{k}),\, \ldots,\, \mathsf{G}_l(\mathsf{k})\}$$

where R denotes some set of random keys. All functions $\mathsf{E}, \mathsf{G}_i, \mathsf{S}_i$ are assumed to output messages of approximately the same length as the keys in R; hence, the update cost can be measured as the number of transmitted messages.

To describe the information that can be recovered from a set of messages M, the *entailment relation* is defined by the following rules:

$$\begin{aligned}
\mathsf{m} \in M \quad &\Rightarrow \quad M \vdash \mathsf{m} \\
M \vdash \mathsf{k} \quad &\Rightarrow \quad M \vdash \mathsf{G}_0(\mathsf{k}), \ldots, \mathsf{G}_l(\mathsf{k}) \\
M \vdash \mathsf{E}_\mathsf{k}(\mathsf{m}), \mathsf{k} \quad &\Rightarrow \quad M \vdash \mathsf{m} \\
\exists I \in \Gamma:\ \forall i \in I:\ M \vdash \mathsf{S}_i(\mathsf{m}) \quad &\Rightarrow \quad M \vdash \mathsf{m}
\end{aligned}$$

By restricting to these relations we essentially assume *secure* encryption and secret sharing schemes. Examples and further comments can be found in [14, Section 3.2]. The set of messages which can be recovered from M using relation $\vdash$ is denoted by $\mathsf{Rec}(M)$.

A *multicast key distribution* protocol for a (static) set of N users and k subsets $S_1, \ldots, S_k \subseteq [N]$ consists of two components – the setup algorithm Setup and an update procedure Update. For simplicity, we assume that each user is member of at least one group.

- Initially, Setup assigns each user $n \in [N]$ a secret key $\mathsf{k}^{(0)}_{\{n\}}$, which is either a uniformly random key from R, or a pseudorandom key that was derived through a sequence of applications of G to another key k'_n, i.e. $\mathsf{k}^{(0)}_{\{n\}} = \mathsf{G}_{l_1}(\mathsf{G}_{l_2}(\ldots(\mathsf{G}_{l_j}(\mathsf{k}'_n))\ldots))$ with $j \geq 0$, where k'_n must not coincide with any of the keys assigned to users in $[N]$. Furthermore, Setup generates a set msgs(0) of so-called *rekey* messages to establish group keys $\mathsf{k}^{(0)}_{S_i}$ (for all $i \in [k]$) among all members of the groups S_i.
- In round t, the algorithm Update takes as input a user identity $n \in [N]$, assigns this user a fresh key $\mathsf{k}^{(t)}_{\{n\}}$ and outputs some rekey messages msgs(t) to establish a fresh group key for each group of which i was a member. For all other members $n' \in [N] \setminus \{n\}$, we set $\mathsf{k}^{(t)}_{\{n'\}} := \mathsf{k}^{(t-1)}_{\{n'\}}$.

For *correctness*, we require that, for any adversarially chosen subgroup structure and any sequence of updating users $(n_1, \ldots, n_t)$, for all $j \in [k]$ each member i of subgroup S_j can recover $\mathsf{k}^{(t)}_{S_j}$, i.e.

$$\mathsf{k}^{(t)}_{S_j} \in \mathsf{Rec}\left(\{\mathsf{k}^{(t)}_{\{n\}}\} \cup \bigcup_{\iota \in [t]_0} \mathrm{msgs}(\iota)\right).$$

For security, we assume the minimal requirement of *post-compromise security*, namely that no group key can be recovered from members outside the group and/or old key material, i.e.

$$\mathsf{k}^{(t)}_{S_j} \notin \mathsf{Rec}\left(\bigcup_{n \in [N] \setminus S_j} \{\mathsf{k}^{(t)}_{\{n\}}\} \cup \bigcup_{\substack{n \in [N],\\ \iota \in [t-1]_0}} \left(\{\mathsf{k}^{(\iota)}_{\{n\}}\} \setminus \{\mathsf{k}^{(t)}_{\{n\}}\}\right) \cup \bigcup_{\iota \in [t]_0} \mathrm{msgs}(\iota)\right).$$

B.2 Key Graphs

The execution of any multicast key distribution protocol can be reflected by a graph structure representing recoverability of the keys involved (cf. [14]). To define this graph, we first need to recall the definition of *useful* keys and messages.

A secret key k is called *useless at time t* if it can be recovered from old key material, i.e. if $\mathsf{k} \in \mathsf{Rec}\left(\bigcup_{\substack{n \in [N],\\ \iota \in [t-1]_0}} \left(\{\mathsf{k}^{(\iota)}_{\{n\}}\} \setminus \{\mathsf{k}^{(t)}_{\{n\}}\}\right) \cup \bigcup_{\iota \in [t]_0} \mathrm{msgs}(\iota)\right)$, otherwise k is called *useful*. If a multicast key distribution protocol satisfies correctness and post-compromise security, then for all $t \in \mathbb{N}$, $n \in [N]$, $j \in [k]$ it must hold that the user's keys $\mathsf{k}^{(t)}_{\{n\}}$ as well as the group keys $\mathsf{k}^{(t)}_{S_j}$ are useful at time t.

To decide whether a message is useful, one has to consider the information it contains, where messages can be arbitrarily nested applications of encryption E and secret sharing S. Thus, a message m is said to *encapsulate* a (pseudo)random key k if $\mathsf{m} = e_1(e_2(\ldots(e_j(k))\ldots))$ where $e_i = \mathsf{E}_{\mathsf{k}_i}$ or $e_i = \mathsf{S}_{h_i}$ (for some key k_i and $h_i \in [h]$). A message is then called *useful* if it encapsulates a useful key.

Definition 3 (Key graph [14])**.** *The* key graph $\mathcal{KG}_t = (\mathcal{V}_t, \mathcal{E}_t)$ *for a multicast key distribution protocol at time t is defined as follows.* $\mathcal{V}_t$ *consists of all the keys that are useful at time t, and* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ *consists of all ordered pairs* $(\mathsf{k}_1, \mathsf{k}_2)$ *such that one of the following is true:*

- *There exists* $j \in [l]$ *auch that* $\mathsf{k}_2 = \mathsf{G}_j(\mathsf{k}_1)$*.*

- *There exists some message $m \in \bigcup_{j\in[t]_0} \mathrm{msgs}(j)$ such that $m = e_1(\mathsf{E}_{\mathsf{k}_1}(e_2(\mathsf{k}_2)))$ for some sequences of encryption and secret sharing e_1 and e_2, and e_2 does not contain any encryption under a key that is useful at time t.*

Edges of the second type are called communication edges*.*

One can show that for any node k in $\mathcal{KG}$ there is at most one edge of the first type incident on k (see [14, Proposition 1] for a proof). Note that edges of the first type do not incur any communication cost; thus, in the following we will be interested the number of communication edges. To this aim, we prove the following properties of key graphs, which in particular show that key-derivation graphs as defined in Section 3.2 are just a special case of key graphs (cf. Definition 1).

Lemma 8. *Consider a secure multicast key distribution protocol for $N \in \mathbb{N}$, $S_1, \ldots, S_k \subseteq [N]$. Then, for any $t \in \mathbb{N}$ and sequence of updates $(n_1, \ldots, n_t)$, the corresponding key graph $\mathcal{KG}_t$ satisfies the following two conditions.*

1. *For $n \in [N]$ and $j \in [k]$ there exist nodes v_n and v_{S_j} in $\mathcal{KG}_t$, and for $n \neq n' \in [N]$ it holds $v_n \neq v_{n'}$.*
2. *For every pair of keys $\mathsf{k}_1, \mathsf{k}_2$ that are useful at time t, such that $\mathsf{k}_2 \in \mathsf{Rec}\left(\{\mathsf{k}_1\} \cup \bigcup_{\iota\in[t]_0} \mathrm{msgs}(\iota)\right)$, there exists a path from k_1 to k_2 in $\mathcal{KG}_t$ that only consists of keys k such that $\mathsf{k} \in \mathsf{Rec}\left(\{\mathsf{k}_1\} \cup \bigcup_{\iota\in[t]_0} \mathrm{msgs}(\iota)\right)$.*

Proof. By definition the keys $\mathsf{k}^{(t)}_{\{n\}}$ for users $n \in [N]$ are distinct and there exists a group key $\mathsf{k}^{(t)}_{S_j}$ for each group S_j ($j \in [k]$). To prove property 1, it remains to prove that these keys are useful, hence represent a node in $\mathcal{KG}_t$. For group keys $\mathsf{k}^{(t)}_{S_j}$ this follows immediately from security of the scheme. For the users' private keys, recall that we assume that for all $n \in [N]$ there exists some $j \in [k]$ such that $n \in S_j$. We assume for contradiction that the user's current key $\mathsf{k}^{(t)}_{\{n\}}$ was useless, i.e. could be recovered from users' old keys that have been replaced in rounds $[t]$ and the rekey messages. But by correctness it must hold that user n can recover the group key $\mathsf{k}^{(t)}_{S_j}$ from its own key and the rekey messages. This, however, would imply that $\mathsf{k}^{(t)}_{S_j}$ can be recovered from old keys and rekey messages, hence $\mathsf{k}^{(t)}_{S_j}$ would be useless – a contradiction.

For the second property, we refer to [14, Lemma 1] for a proof. □

Note that the converse of property 2 is not true, since e.g. a message $\mathsf{E}_{\mathsf{k}_1}(\mathsf{S}_1(\mathsf{k}_2))$ with useful keys $\mathsf{k}_1, \mathsf{k}_2$ incurs an edge $(\mathsf{k}_1, \mathsf{k}_2)$ while k_2 can only be recovered from k_1 if $\{1\} \in \Gamma$.

B.3 Lower Bound on the Average Update Cost

The communication complexity of a multicast encryption scheme after t updates is given by $\left|\bigcup_{\iota\in[t]_0} \mathrm{msgs}(\iota)\right|$. To measure the efficiency of the protocol we will consider the *amortized communication complexity*

$$\mathrm{Com_A} := \Big|\bigcup_{\iota\in[t]_0} \mathrm{msgs}(\iota)\Big|/t \ .$$

We now are ready to compute a bound on the expectation of $\mathrm{Com_A}$ in the scenario where in every round the updating party is chosen uniformly at random. The result improves on [14, Theorem 1] in two aspects. It generalizes the bound to the setting of several potentially overlapping groups, and further gives a bound on the *average* communication complexity of updates opposed to a worst case bound.

Theorem 4. *Consider a multicast key-distribution protocol for $N \in \mathbb{N}$, $S_1, \ldots, S_k \subseteq [N]$ that is secure in the symbolic model. Then the expected amortized average communication cost after t updates is bounded by*

$$\mathbb{E}[\mathrm{Com_A}] \geq (1 - 1/t) \cdot \frac{1}{N} \sum_{\emptyset \neq I \subseteq [k]} |P_I| \cdot \log(|P_I|) \ .$$

and the asymptotic update cost of the protocol is at least $\frac{1}{N} \sum_{\emptyset\neq I\subseteq[k]} |P_I| \cdot \log(|P_I|)$.

Proof. We prove the result by showing that the average communication complexity after the tth update has size at least $(t-1)\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$. To this end, we will show that with every update on average at least $\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$ useful messages become useless.

Let $1\leq t'\leq t$. Consider the useful nodes v_n guaranteed to exist by Lemma 8, Property 1 after the $(t'-1)$st update (where the 0th update is to be understood as Setup). We show that the key graph $\mathcal{KG}_{t-1}$ contains a subgraph $\mathcal{G}'_{t-1}$ which satisfies the requirements of Lemma 3:

By Lemma 8, Property 2, for each $n\in[N]$ and $j\in[k]$ such that $n\in S_j$ there exists a path $\mathcal{P}_{n,j}$ from v_n to v_{S_j} in $\mathcal{KG}_{t-1}$ such that all keys associated to nodes in $\mathcal{P}_{n,j}$ can be recovered from $\mathsf{k}_{\{n\}}^{(t-1)}$ and the sent messages. Let $\mathcal{G}'_{t-1}$ denote the union of these paths. It remains to argue that all nodes v_n are sources in $\mathcal{G}'_{t-1}$. For contradiction, assume there exists $n,n'\in[N]$, $j\in[k]$ such that $n'\in\mathcal{P}_{n,j}$. But Update only replaces one user's private key; thus, if in the next round an update for n was generated, only n's private key would be replaced, but not n''s. Hence, security would be broken because n''s current key $\mathsf{k}_{\{n'\}}^{(t)}=\mathsf{k}_{\{n'\}}^{(t-1)}$ can be recovered from n's old key, and by correctness, for any $j'\in[k]$ such that $n'\in S_{j'}$, the key $\mathsf{k}_{S_{j'}}^{(t)}$ can be recovered from $\mathsf{k}_{\{n'\}}^{(t)}$. This proves that $\mathcal{G}'$ indeed satisfies the properties of Lemma 3.

Now, recall, that at most one of the edges incident to a node in the key graph is not a communication edge. Thus, by Lemma 3 the number of useful messages encapsulating keys that can be reached from v_n is on average at least $\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$.

Note that one of the keys v_n becomes useless after the t'th update. By Lemma 8, Property 2 all other nodes in $\mathcal{D}(v_n)\subseteq\mathcal{G}'_{t-1}$ and in turn messages encapsulating descendants become useless as well. With the argument above we obtain that with the t'th update on average at least $\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$ messages become useless. By linearity of expectation and since useless messages never become useful again this implies that after the t'th update on average at least $(t-1)\frac{1}{N}\sum_{\emptyset\neq I\subseteq[k]}|P_I|\cdot\log(|P_I|)$ messages have been sent. Now dividing by t yields the claim. □