The Art of Bonsai: How Well-Shaped Trees Improve the Communication Cost of MLS*

Céline Chevalier * †, Guirec Lebrun*‡, Ange Martinelli‡, Jérôme Plût‡
*DIENS, École normale supérieure, CNRS, PSL University, INRIA, Paris, France
† CRED, Paris-Panthéon-Assas University, Paris, France
‡ ANSSI, Paris, France

***Abstract*—Messaging Layer Security (MLS) is a Secure Group Messaging protocol that uses for its handshake a binary tree – called a Ratchet Tree – in order to reach a logarithmic communication cost in the number of group members. This Ratchet Tree represents users as its leaves; therefore any change in the group membership results in adding or removing a leaf in the tree. MLS consequently implements what we call a tree evolution mechanism, consisting of a user add algorithm – determining where to insert a new leaf – and a tree expansion process – stating how to increase the size of the tree when no space is available for a new user. The tree evolution mechanism currently used by MLS is designed so that it naturally left-balances the Ratchet Tree. However, such a tree structure is often quite inefficient in terms of communication cost. Furthermore, one may wonder whether the binary Ratchet Tree has a degree optimized for the features of MLS.**

Therefore, we study in this paper how to improve the communication cost of the handshake in MLS – realized through an operation called a commit – by considering both the tree evolution mechanism and the tree degree used for the Ratchet Tree. To do so, we determine the tree structure that optimizes its communication cost and we propose algorithms for both the user add and the tree expansion processes, that allow to remain close to that optimal structure and thus to have a communication cost as close as possible to the optimum. We also find out the Ratchet Tree degree that is best suited to a given set of parameters induced by the encryption scheme used by MLS. This study shows that when using classical (i.e. pre-quantum) ciphersuites, a binary tree is indeed the most appropriate Ratchet Tree; nevertheless, with post-quantum algorithms, it generally becomes more interesting to use instead a ternary tree.

Our improvements do not change the TreeKEM protocol and are easy to implement. With parameter sets corresponding to practical ciphersuites, they reduce TreeKEM's communication cost by 5 to 10%. In particular, the gain of 10% appears in the post-quantum setting – when both an optimized tree evolution mechanism and a ternary tree are necessary –, which is precisely the context where any optimization of the protocol's communication cost is welcome, due to the large bandwidth of PQ encrypted communication.

***Index Terms*—MLS, TreeKEM, CGKA, Key Tree, Binary Tree, Ternary Tree**

This paper appears in the proceedings of IEEE EuroS&P 2025.

1. Introduction

With the development of Secure Messaging protocols that enable end-to-end encrypted communication between two users, the need for a dedicated Secure Group Messaging protocol has arisen in the last few years. Indeed, the group conversation functionality offered in practice by (point-to-point) Secure Messaging applications relies on *ad hoc* constructions that either lower the protocol's security level – such as the SenderKey protocol, used e.g. by WhatsApp [1] – or that are quite inefficient – like the Pairwise protocol of Signal application –, with a communication cost scaling linearly with the number n of group members.

Messaging Layer Security (MLS) [2] is an IETF standard, released in July 2023, for a Secure Group Messaging protocol that is designed to keep the security properties of point-to-point Secure Messaging protocols, including Forward Secrecy (FS) and Post-Compromise Security (PCS) (cf. Appendix A.1), with a communication cost growing logarithmically with the number of users: $O(\log_2(n))$.

For this purpose, MLS uses as core component a group key exchange mechanism – conceptualized by [3] as a Continuous Group Key Agreement (CGKA) – named TreeKEM, that provides the group with a common group key that securely evolves over time in order to maintain the aforementioned forward secrecy and post-compromise security, despite the changes in the group membership.

1.1. Optimizing Group Communication with a Key Tree

1.1.1. TreeKEM's Ratchet Tree. In order to reach the aimed communication cost of $O(\log_2(n))$, TreeKEM relies on a binary tree called a Ratchet Tree, whose leaves represent the group members and where all nodes are associated with encryption key-pairs that are hierarchically organized: each key-pair is deterministically generated from a secret seed that is directly derived – through a key derivation process – from one of the seeds associated with the two children nodes located beneath. A central concept in TreeKEM, called the tree invariant, is that any user (leaf) in the tree knows all the secret elements (secret seeds and secret encryption keys) from the nodes above them, and only them. Consequently, the secret seed at the root of the tree is the only secret element known by all

users; the group key is directly derived from that common value.

The idea behind a key tree such as TreeKEM's Ratchet Tree is that the rotation of the group key, initiated after the arrival or departure of a group member or the update of a user's keying material, and occurring in a process named a commit, impacts the seeds and keys of all the nodes located above that updater, up to the root, in what is called its direct path. Consequently, the public and private information[1] that needs to be transmitted to the group is proportional to the number of nodes in the updater's direct path. In a binary tree, the average length of a user's direct path is in $O(\log_2(n))$, which corresponds to the expected logarithmic communication cost of that CGKA.

1.1.2. Influence of the Tree Structure. In practice, the lower bound of $\log_2(n)$ is rarely reached. This ideal value indeed corresponds to the cost of a perfect binary tree, i.e. a tree whose nodes all have exactly two children and where the leaves are all located at the same level.

Not only is this ideal tree impossible when the number of users is not a power of two, but for a given number n of users, the tree structure, i.e. the unordered arrangement in space of the leaves, also has a significant influence on the communication cost. The structure of a CGKA's Ratchet Tree is determined by the way the protocol adapts this tree to the natural evolution of the group membership, with user departures at arbitrary locations and user arrivals at locations specified by the protocol. A CGKA's tree evolution mechanism therefore consists of the following algorithms[2]:

- a user add algorithm, which specifies where to add a new user among all the available locations;
- a tree expansion algorithm that increases the size of the tree when a new user has to be added to the group and there is no free space available for it, and that decreases the tree size whenever it is possible.

TreeKEM's user add algorithm inserts new users at the leftmost available location in the tree. When needed, it increases the Ratchet Tree to the right, by creating a new root above the current one and by adding the extra leaves in a new subtree attached to the right of that new root. This mechanism left-balances TreeKEM's Ratchet Tree, by grouping most leaves on the left side of the tree.

However, we show in Section 3 that a left-balanced structure does not minimize the communication cost of a CGKA, and consequently, that the tree evolution mechanism used by TreeKEM is not optimal in that matter.

1.1.3. Influence of the Tree Degree. The degree, or arity, of a tree is the maximum number of children that any node in the tree is allowed to have. The lower bound of $\log_2(n)$ stems from the fact that TreeKEM uses a binary tree as its Ratchet Tree. This optimal bound varies according to the tree degree: the higher the degree, the shorter the average direct path of a user becomes, which decreases the number of updated public keys that have to be broadcast to the entire group.

However, this advantage comes at a cost. For instance, in CGKAs using Diffie-Hellman (DH) trees (cf. Section 1.3), a tree of degree m implies that any internal node's secret must be computed with a key agreement relying on the m-party generalized Group Diffie-Hellman problem, which is more costly – computationally speaking as well as regarding the bandwidth consumption – than a standard 2-party DH key agreement.

In the case of TreeKEM, increasing the tree degree also increases the number of recipients to whom the encrypted secret seeds associated with the nodes of the updater's direct path must be transmitted. This number of recipients corresponds to the number of nodes in the updater's copath, i.e. the number of sibling nodes of this user and of its ancestors (cf. Section 2.2).

The optimal tree degree for TreeKEM is therefore the one which offers the best trade-off between the number of nodes in a user's direct path and the number of nodes in its copath, for all users in the tree. However, to the best of our knowledge, no study has been carried out to find out precisely the optimal degree for TreeKEM.

1.2. Our Contributions and Outline of this Paper

We determine in this paper the way to optimize the communication cost of TreeKEM by considering the two factors of tree structure and tree degree detailed above.

Our focus on the communication cost comes from the fact that the existing post-quantum encryption schemes, that any key agreement protocol must be able to implement – in the current context of post-quantum hybridization – while remaining efficient, increase much more the bandwidth than the computational and memory costs. The communication cost therefore represents the bottleneck for the efficiency of the protocol.

We study in Section 3 the optimization of a binary tree, as already used by TreeKEM for its Ratchet Tree. To do so, we give in Section 3.1 a simple mathematical model of the communication cost of a commit in TreeKEM. We prove that our metric accurately models that communication cost under the hypothesis that the commit messages originate from uniformly distributed users in the user tree. We then use that metric to determine in Section 3.2 the optimal structure of a full[3] tree of an arbitrary degree $m \geq 2$, i.e. the one that minimizes the communication cost of a commit in a group of n users. In Section 3.4 we propose an optimized tree evolution mechanism, based on simple, yet effective, user add and tree expansion algorithms that are applicable to trees of any degree m. We then experimentally show in Section 3.5, thanks to a dedicated implementation, that the communication cost of our optimized method is much closer to the optimal cost than the one of TreeKEM's method. Our results for different parameters are depicted in Table 1.

In Section 4, we consider the influence of the tree degree on the communication cost of TreeKEM, with respect to our metric, in the most general case of *non-full* trees

[1] The public elements mentioned here are the updated public keys and the private ones are the secret seeds related to that public keys.

[2] The case of a user removal is not considered in a tree evolution mechanism, since the protocol does not control the departure of users from the group and thus cannot choose which leaves to remove from the Ratchet Tree.

[3] As detailed later, a *full* tree of degree m is a tree where any node has either zero or m children.

that corresponds to the feature of real Ratchet Trees. We prove that binary and ternary (non-full) trees are the most efficient Ratchet Trees in the most common use cases[4]. We also compare, theoretically and experimentally, these two degrees and determine a bound for the communication ratio λ – defined in Section 2.3.2 – that separates their respectives areas of optimality and underlines that for most post-quantum ciphersuites, it is more relevant to use a ternary tree than a binary one.

Our experimental results show that in that context, the combination of using our optimized tree evolution algorithms and replacing the current binary tree with a ternary one, brings an average gain of around 10% compared to the current implementation of TreeKEM (cf. Table 2), at the expense of only few additional computations demanded by the optimized algorithms. This gain is highly appreciable since it occurs in the post-quantum framework, which already needs a large bandwidth and thus for which any improvement is welcome.

1.3. Background on Key Trees

The idea to use key graphs in order to decrease the communication cost of a group key exchange goes back long before MLS: the seminal concurrent works of [4] and [5] introduce that concept – called Logical Key Hierarchy – in the late 1990s, in the context of a centralized key distribution system, where a key server manages the creation and distribution of all keys in the graph.

In the decentralized framework, where peers mutually interact without any central authority in order to generate a common group key, [6], followed by the ELK protocol [7], proposes a new tree-based architecture called One-Way Function Tree (OFT) in which any node key is no longer randomly generated by a central entity, but can be deterministically computed, with the use of symmetric primitives such as pseudo-random functions, from the keys belonging to that node's children.

1.3.1. Diffie-Hellman Trees. The seminal works of [8] (NAGKA) and [9] (TGDH) pave the way to Diffie-Hellman trees by combining the concepts of children-dependent key graph and (2-party) DH key agreement. A number of subsequent papers continue in this vein, [10], [11], [12], [13] among others. However, ART [14] is the first fully asynchronous DH-tree-based group key agreement protocol, that no longer requires all users to be online for every group operation and additionally provides the expected security property of post-compromise security. This protocol has been chosen as MLS's CGKA in its initial IETF RFC draft [15].

1.3.2. The TreeKEM Protocol. TreeKEM [16] is a versatile CGKA which uses a binary tree and considers the cryptographic primitives – and in particular, a public-key encryption scheme (PKE) – as black boxes. Consequently, it is not linked to any particular cryptographic assumption and offers the crypto-agility needed to allow the replacement of any untrusted primitive. This protocol constitutes MLS's CGKA since version two of its RFC draft [17], until the current IETF standard [2].

1.3.3. Tree Degree Variations. Within the large body of literature devoted to tree-based group key agreement, only few papers seem to have questioned the use of binary trees for this type of protocol, notably because degree 2 is best suited for the numerous protocols relying on the (2-party) Diffie-Hellman problem. The early work of [5] studies the best key graph in the framework of centralized Logical Key Hierarchy, and reaches the conclusion that a rooted-tree of degree 4 is the best architecture for their Secure Group. Their conclusion is however orthogonal to our present study, since their analysis is based on the computational cost of the protocol and not on the communication one.

The main alternate propositions to binary trees are protocols using the Bilinear Diffie-Hellman problem [18], [19], after a pairing-based three-party DH variant has been proposed by [20]. For its part, [21] uses, inside a ternary tree, the GDH.2 protocol of [22] based on the generalized Group Diffie-Hellman problem, and shows that this architecture is more efficient in terms of communication cost than a binary Diffie-Hellman tree. [23] extends the latter work by allowing a number of group members different from a power of three, using non-full ternary trees. No analysis is however carried out regarding the most efficient structure of that tree.

Regarding TreeKEM, the authors [16] specify that their protocol works with trees of any degre; however it has been used by MLS with binary trees only.

On the other edge of the degree spectrum, [24] proposes a novel CGKA called Chained-CmPKE, which uses an augmented encryption scheme ("multi-recipient PKE") in order to have an efficient distribution of the handshake messages in a *comb* tree, i.e. in a tree of unitary height and degree n for n users.

2. Preliminaries: Tree used in TreeKEM

2.1. Notations and Terminology

The output of a probabilistic algorithm is represented by $\leftarrow$ and that of a deterministic algorithm is given by $:=$. The notation $\cdot||\cdot$ is used for the concatenation operation. $\cdot|\cdot$ represents the *or* operator. $|\mathcal{S}|$ denotes the cardinality of a set $\mathcal{S}$. The operators $\lfloor\cdot\rceil$ and $\lceil\cdot\rceil$ respectively denote the rounding and ceiling values of a decimal number. Without any base explicitly indicated, $\log(\cdot)$ denotes the logarithm in base 2.

2.2. Dendrologic Definitions

2.2.1. Trees Used in a CGKA. The trees considered in this paper are *rooted trees*, which we shall always represent graphically with the root on top. We use two types of trees, both of them defined recursively:

1) *plane trees* [25, I.5.1] are defined as either a leaf, designated by the symbol ℓ, or as an ordered tuple $(B_1, \ldots, B_m)$ of trees called its branches;
2) *non-plane trees* [25, I.5.2] are either a leaf ℓ or an unordered set $\{B_1, \ldots, B_m\}$ of trees.

[4] All standard encryption schemes that we have considered are indeed adapted to these two tree degrees, except for the HPKE ciphersuite based on the post-quantum KEM ClassicMcEliece.

In boths cases, we call a tree *irreducible* [26, 3.3] if no node has a single child (which corresponds to excluding the case $m = 1$ from the definitions above). Nodes with a single child are redundant in a Ratchet Tree[5] and therefore we shall only consider irreducible trees.

TreeKEM uses plane trees, where the order of the leaves corresponds to the users' indices. However, since horizontally permuting the Ratchet Tree's branches has no impact on that tree's communication cost, the theoretical study in Section 3 and Section 4 is done on non-plane trees.

2.2.2. Tree Features. We define below the basic features characterizing trees.

Weight. The weight $w(T)$ of a tree T is the number of its leaves.

Node and Tree Degrees. The degree $deg(v)$ of a tree node v is the number of branches immediately below that node (and zero for a leaf). The degree of a tree T is the maximum degree m of any node of the tree (in which case that tree is said to be m-ary).

Height. The height $h(T)$ of a tree T is defined recursively: it is zero for a leaf, and otherwise one plus the maximal height of all branches. It is also the maximal length of a path from the root to a leaf.

Node Depth. The depth $d(v)$ of an internal node v is the length of the path from the tree root to that node. The root itself has depth zero, while the lowest leaves in the tree have a depth equal to the tree height.

Full and Perfect Trees. A tree is called full of degree $m \geq 2$ when each node has either zero or m children. In particular, all full trees are irreducible. A perfect m-ary tree is a full tree whose leaves are all located at the same depth : in that case, its weight is m^h, where h is its height.

2.3. TreeKEM's Ratchet Tree

A detailed description of how TreeKEM – as standardized in RFC 9420 [2] – works as a CGKA protocol, is given in Appendix A. We focus hereunder on the binary tree used by TreeKEM.

2.3.1. Ratchet Tree. As stated above, the key tree used by TreeKEM to perform its group key agreement – called the Ratchet Tree – is a full binary rooted tree where users are represented by the leaves and the group key is computed at the root. Each node of this Ratchet Tree, except for the root, is associated with a local state γ which notably includes an encryption key-pair. This key-pair is issued from a seed called a path secret, which is itself derived from the one of the node's children. The full description of a node state is provided in Appendix A.

Formal and Logical Tree Representations. For practical reasons, the MLS specifications represents a Ratchet Tree as a perfect binary tree of height h, for all numbers of users $n \leq 2^h$ and whatever their relative locations in the tree. Therefore, when n is not a power of two, some of the 2^h leaves are not associated with any group member and thus have an empty state. Such leaves are called blank leaves. Moreover, as TreeKEM's Ratchet Tree is irreducible, no internal node can have a single child. Consequently, any internal node that has a blank child with all this child's descendants blank, is itself blanked.

Since their state is empty, blank nodes do not take part in TreeKEM's operations until they are filled again. A perfect tree with blank nodes can thus be logically represented by a single non-perfect tree where blank nodes are removed and where leaves that previously had blank ancestors are attached higher in the tree (cf. Figure 1). The latter representation is called logical representation whereas the architecture specified in MLS standard is named formal representation and corresponds to the practical implementation of that tree[6].

The use of blank nodes with the formal representation of TreeKEM's Ratchet Tree requires to adapt the concepts of direct path and copath of a node (cf. below). This is done *via* the notion of resolution, detailed beneath:

Resolution of a Node (from [3]). The resolution of a node v from a tree is a set of nodes defined as follows:

- if v is a non-blank node, then $res(v) = \{v\}$;
- if v is a blank leaf, then $res(v) = \varnothing$;
- if v is a blank internal node, then: $res(v) = \cup_{v' \in Children(v)} res(v')$.

(Filtered) Direct Path and Copath of a Node. The direct path of a node v, noted $\mathcal{P}_v$, is composed of all the ancestors of that node, up to the root. The copath $\mathcal{CP}_v$ of this node contains its sibling(s) and the ones of its ancestors (i.e. of the nodes belonging to its direct path).

In the formal tree representation, a node's direct path may include blank nodes, which must not be taken into account. Consequently, the MLS standard [2] defines the notion of a node's filtered direct path, which is that node's direct path from which all nodes that have a child with an empty resolution are removed. The filtered direct path corresponds in the formal tree representation to the direct path used in logical representation.

In the remainder of the document we will freely use the term direct path to refer to both concepts.

2.3.2. Ratchet Tree Evolution: the Commits. The update of TreeKEM's Ratchet Tree, in order to follow the evolution of the group membership as well as to periodically refresh the users' keying material, is realized *via* an operation named commit, which groups together and validates proposals for change sent by any group member. This operation is carried out by a randomly selected user called the committer, here noted u_c.

As shown in the detailed description of a commit, in Appendix A, a commit refreshes the committer's direct path by updating the path secret of all nodes in its direct path. These refreshed path secrets must then be encrypted and sent to all users that are located beneath these nodes. Figure 15 depicts this update process.

[5]Indeed, internal nodes in a Ratchet Tree contain intermediate encryption keys common to a subgroup of leaves. When a node has only one child, the encryption keys of that node and of its child are related to the same subgroup of leaves beneath them and are thus redundant.

[6]We underline that a single logical representation may be associated to several formal representations, the converse being false, because the formal representation is more precise regarding the leaf arrangement than the logical one.

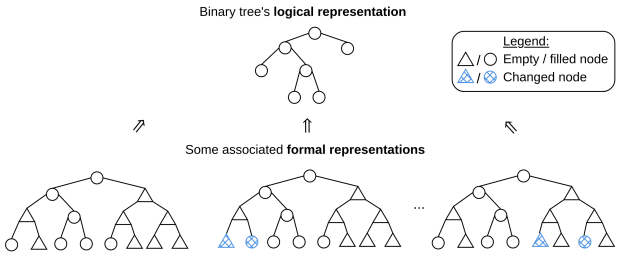


Figure 1: Correspondence between a tree's logical representation and some of the various formal representations associated with it.

Broadcast of a commit message to the Ratchet Tree. All information related to the commit is grouped within a single commit message C that is broadcast to the group by the committer u_c, and that consists of:

- the list of proposals that the commit implements ($\vec{P}$);
- the updated public local state γ_c^{pub} of the committer;
- the new public encryption keys $(pk_{v_p})_{v_p \in \mathcal{P}_c \setminus \{v_{root}\}}$ from the committer's direct path $\mathcal{P}_c$ (except the root node v_{root} that has no public key);
- the path secrets ps_{v_p} of the nodes $v_p \in \mathcal{P}_c$ from the committer's direct path, each encrypted under the public key of v_p's child v_{ch} on the committer's copath $\mathcal{CP}_c$[7].

$$C(u_c) = \vec{P} \;||\; \gamma_c^{pub} \;||\; \left(pk_{v_p}\right)_{v_p \in \mathcal{P}_c \setminus \{v_{root}\}} \;||\; \left(\mathrm{Enc}(pk_{v_{ch}}, ps_{v_p})\right)_{v_p \in \mathcal{P}_c} \quad (1)$$

Encryption Process. MLS states that the encryption of private messages is carried out through the Hybrid Public Key Encryption (HPKE) paradigm [27]: a Key Encapsulation Mechanism (KEM) generates a shared secret that is used as key and nonce for an AEAD[8]-based symmetric encryption of the plaintext.

The ciphertext size $|ct_m|$ generated by an HPKE scheme thus depends on the plaintext size $|m|$, on the size of the AEAD tag and on the KEM's features. In this paper, we focus on the encryption of the path secrets that must be transmitted within a commit.

3. Optimizing a Tree Structure

We study in this part the influence of a Ratchet Tree's structure, i.e. the disposition of its leaves, on its communication cost. With a cost metric defined in Section 3.1, we show in Section 3.2 that for any degree, a full tree is optimal only if it is depth-balanced. Section 3.3 shows that the tree evolution mechanism currently used in TreeKEM is not optimal regarding this cost function, and we propose in Section 3.4 optimized tree evolution algorithms to improve the efficiency of that CGKA, which we experimentally compare to TreeKEM's.

[7]MLS standard blanks all the nodes in the direct path of a leaving member. Consequently, until these internal nodes are one-by-one filled again by subsequent path updates initiated by the neighbors of the removed user, we may have in the tree several leaves joined together by a blank node, which corresponds, in logical representation, to attaching these leaves higher in the tree, and sometimes even directly to the root. In this case, we may need to encrypt several times the same path secret. However, as this structure is only temporary, we do not take it into account in our study and we consider that a path secret is only encrypted as many times as the node has children in the committer's copath (once in the case of a binary tree).

[8]Authenticated Encryption with Associated Data

3.1. Communication Cost Metric

3.1.1. Tree-Dependent Cost. In this part we model the communication cost of a Ratchet Tree and isolate the parts of this cost which can be associated to the shape of the tree. Since the various proposal messages are broadcast to all users, their impact on the communication cost is independent from the tree structure. On the other hand, the commit message from Equation (1) includes the lists of updated public keys and encrypted path secrets from the committer's direct path. Both of those values depend on the Ratchet Tree's structure.

From Equation (1) we see that the size of a commit message C_i originated by the i-th user is:

$$|C_i| = |pk| \cdot |\mathcal{P}_i| + |ct| \cdot |\mathcal{CP}_i| + z_i \quad (2)$$

where $z_i = \left|\vec{P_i}\right| + \left|\gamma_i^{\mathrm{pub}} \backslash pk_i\right|$. The exclusion of the term pk_i from z_i is due to its inclusion in the term $|pk| \cdot |\mathcal{P}_i|$[9].

The size of the term z_i is independent from the user's position. On the other hand, the two other terms depend on the user's position as well as on the cryptosystem used by the CGKA, through the sizes of public keys $|pk|$ and of ciphertexts $|ct|$.

The dependency on the underlying cryptosystem can actually be controlled by a single parameter, as follows:

Definition 1 (Communication ratio). For a KEM $\mathcal{K}$, an AEAD scheme $\mathcal{E}$ yielding a tag of size $|tag|$ and a given path secret size[10] $|ps|$, we define the communication ratio λ as:

$$\lambda^{hpke}_{\mathcal{K},\mathcal{E},|ps|} = \frac{\left|pk^{hpke}\right|}{\left|ct^{hpke}_{|ps|}\right|} = \frac{\left|pk^{\mathcal{K}}\right|}{|ct^{\mathcal{K}}| + |tag^{\mathcal{E}}| + |ps|} \quad (3)$$

The values of λ for the (classical) ciphersuites advised by MLS standard, as well as for the post-quantum ciphersuites that are most likely to be used, are detailed in Table 3 in Appendix B.

We work under the hypothesis that the committers are randomly chosen among all users, which is precisely the case in the MLS standard, where the distinction between administrators and other users is left at the applicative level. The influence of the Ratchet Tree in a CGKA where administrators – with the right to commit – are precisely determined, is an open problem out of the scope of our study.

From this hypothesis we can find the average commit cost (up to a constant factor) by summing Equation (2) over all users i. However, the presence in Equation (2) of the direct path and of the co-path of i naturally lead to the following definitions:

[9]The fact that no public key is associated with the Ratchet Tree's root node – which is yet in the committer's direct path – is compensated for by the inclusion of the committer's own public key in the list of public keys.

[10]The size of a path secret is defined by MLS standard as the output of the Extract stage of the Key Derivation Function used within the selected ciphersuite. This value depends on the desired security level.

Definition 2 (Parent cost and sibling cost). We define the parent cost $c_p(T)$ of a tree T as the sum of the length of the paths of all the leaves in the tree, and its sibling cost $c_s(T)$ as the sum of the length of all the copaths.

Equivalently, the parent cost and sibling cost functions may be defined recursively:

$$\begin{aligned} c_p(\ell) &= c_s(\ell) = 0 \\ c_p(T = \{B_1, \ldots, B_m\}) &= w(T) + \sum\nolimits_i c_p(B_i) \\ c_s(T = \{B_1, \ldots, B_m\}) &= (m-1)w(T) + \sum\nolimits_i c_s(B_i) \end{aligned} \tag{4}$$

Using these definitions together with Equation (2), we see that the global commit cost for the tree T is:

$$c(T) = \sum\nolimits_i |C_i| = (c_p(T) \cdot \lambda + c_s(T))\,|ct| + \sum\nolimits_i z_i \tag{5}$$

The term $\sum_i z_i$ is a constant independent from the tree shape, as is the factor $|ct|$. Therefore, we focus on the tree-dependent expression $c_p(T)\lambda + c_s(T)$:

Definition 3 (Tree-Dependent Cost). The tree-dependent cost κ of a tree T is defined as the following function of the parameter λ:

$$\kappa(T) = c_p(T) \cdot \lambda + c_s(T) \tag{6}$$

This function accurately models the two following factors:

- the number of public keys that are refreshed in the committer's direct path (which is captured in the parent cost $c_p(T)$), and the number of encrypted path secrets from that direct path that are sent to the committer's copath (which is represented by $c_s(T)$);
- the influence of the underlying cryptosystem on those two values, expressed through the communication ratio λ.

3.1.2. Impact of Optimizing the Tree-Dependent Cost. Since this paper focuses on the optimization of the tree-dependent cost κ of TreeKEM, we must find out whether this enhancement remains impactful even when taking into account the tree-independent communication cost of a commit.

The latter depends on the primitives used (classical or post-quantum) and on the number and types of proposals. According to the specifications of the MLS standard, the proposal messages have sizes ranging from 52 bytes (for a user remove) to 3.2 kB (for a post-quantum user add). The tree-dependent cost of a commit also depends on the classical or PQ framework and grows linearly with the RT height.

Therefore, the tree-dependent cost in a TreeKEM commit with a single proposal represents 14 to 83% (user add), 32 to 97% (user remove) and 15 to 88% (user update) of the commit message, depending on the parameters used. In a handshake with several proposals, this proportion decreases; however, in many cases, the tree-dependent cost appears influential enough so that our improvement remains significant even considering the total communication cost.

3.2. Optimal Structure of a Full m-ary Tree

3.2.1. Communication Cost of a Full m-ary Tree. In a full tree of degree $m \geq 2$ (noted T_m^f), each node except the root has, by definition, exactly one parent and $m-1$ siblings.

Therefore, the number of nodes in the copath of each leaf is exactly $(m-1)$ times the number of nodes in its direct path. By summing over all leaves, we find that:

$$T_m^f : \quad c_s = (m-1)c_p \Rightarrow \ \kappa = c_p(\lambda + m - 1) \tag{7}$$

In particular, the full binary tree T_2^f has $m = 2$, so that:

$$T_2^f : \quad \kappa = c_p(\lambda + 1) \tag{8}$$

This shows that optimizing the communication cost for a given full tree T^f is equivalent to minimizing its parent cost $c_p(T^f)$.

3.2.2. Depth-Balance of a Full Tree. The size of a commit message is directly dependent of the position of the commiter in the tree. Indeed, in an unbalanced tree, the leaves that have the lowest depth (i.e. that are the closest to the root) have a short direct path which is not representative of the whole tree. Figure 2 compares the values of c_p for a totally unbalanced binary tree (left), which has a maximized height $h_{unbal} = n - 1$, and for a perfect binary tree that minimizes its height ($h = \log_2(n)$).

Definition 4 (Depth-Balanced Full Tree). We say that full tree T^f is depth-balanced if the difference of depth between its higher and lower leaves is lesser than or equal to one: $d_{max}(T^f) - d_{min}(T^f) \leq 1$.

Theorem 1. Let T_m^f be a *full* m-ary tree with a given number of leaves n. Then T_m^f has an optimal tree-dependent communication cost κ^{opt} if and only if it is depth-balanced.

Proof 1. Let x_i be the number of leaves at depth i for $i \geq 1$. From the tree of height h being full and m-ary and the total number of leaves being n, we get the two equations:

$$\sum\nolimits_{i=1}^{h} m^{-i} x_i = 1 \text{ and } \sum\nolimits_{i=1}^{h} x_i = n \tag{9}$$

This allows writing, for any index $k \in [\![2, h]\!]$, the two values x_{k-1} and x_k in terms of the other x_i as follows (with $\sum'$ denoting the sum over all $i \notin \{k-1, k\}$):

$$\begin{aligned} \begin{pmatrix} 1 & 1 \\ m^{1-k} & m^{-k} \end{pmatrix} \cdot \begin{pmatrix} x_{k-1} \\ x_k \end{pmatrix} &= \begin{pmatrix} n - \sum' x_i \\ 1 - \sum' m^{-i} x_i \end{pmatrix} \\ \begin{pmatrix} x_{k-1} \\ x_k \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ m^{1-k} & m^{-k} \end{pmatrix}^{-1} \cdot \begin{pmatrix} n - \sum' x_i \\ 1 - \sum' m^{-i} x_i \end{pmatrix} \\ &= \frac{1}{m-1} \begin{pmatrix} -1 & m^k \\ m & -m^k \end{pmatrix} \cdot \begin{pmatrix} n - \sum' x_i \\ 1 - \sum' m^{-i} x_i \end{pmatrix} \end{aligned} \tag{10}$$

The parent cost of such a tree is $c_p = \sum i x_i$, which can be rewritten, using Equation (10), as:

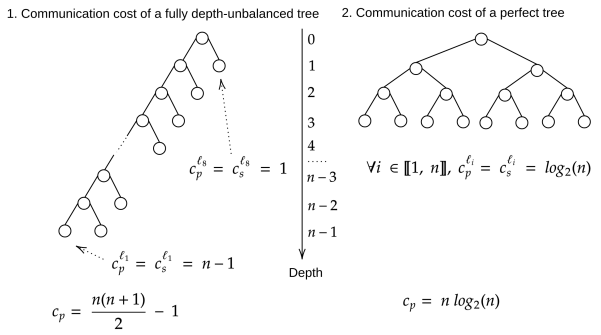


Figure 2: Comparison of the parent costs of a full binary tree bearing n leaves, with unbalanced (left) and balanced (right) structures.

$$\begin{aligned} c_p &= (k-1)x_{k-1} + kx_k + \sum{}' ix_i \\ &= (k + \frac{1}{m-1}, \frac{-m^k}{m-1}) \cdot \begin{pmatrix} n - \sum{}' x_i \\ 1 - \sum{}' m^{-i} x_i \end{pmatrix} \\ &\qquad + \sum{}' ix_i \\ &= nk + \frac{n-m^k}{m-1} \\ &\qquad + \sum{}' \frac{((m-1)(i-k) - 1 + m^{k-i})x_i}{m-1} \\ &= nk + \frac{n-m^k}{m-1} + \frac{1}{m-1} \sum{}' c_{k-i} x_i \end{aligned} \tag{11}$$

where $c_j = m^j - (m-1)j - 1$. Since $c_j = 0$ when $j \in \{0, 1\}$ and $c_j > 0$ otherwise, the minimal value of the parent cost c_p is reached when $x_i = 0$ for $i \notin \{k, k-1\}$. Furthermore, the only possibility for $x_{k-1}, x_k \geq 0$ is $n \leq m^k \leq mn$, which ensures that $k = \lceil \log_m(n) \rceil = h$. This proves that a full tree has an optimal cost only when its leaves are located at depths h and $h-1$, which corresponds to a depth-balanced tree. □

In particular, perfect trees are exactly depth-balanced trees where the number of leaves is a power of m.

The optimal parent cost of a (depth-balanced) full tree is given by Equation (11), with the term $\sum{}' c_{h-i} x_i = 0$ and $k = h$. Therefore, we have:

Lemma 1 (Optimal Communication Cost of a Full m-ary Tree). The optimal (i.e. minimal) communication cost of a full m-ary tree T_m^f, of height $h = \lceil \log_m(n) \rceil$ and bearing n leaves, is:

$$\kappa^{\text{opt}}(T_m^f) = \left(hn + \frac{n - m^h}{m-1} \right) (\lambda + m - 1) \tag{12}$$

3.3. Efficiency of TreeKEM's Binary Ratchet Tree

As stated above, TreeKEM's tree evolution mechanism adds a new user to the group by filling the leftmost blank leaf in the tree. In the case of a full tree, it increases the tree size by adding a new root on top of the current one and attaching to the right of the latter a blank subtree whose leaves are then filled, from left to right, by the additional user(s). The tree is reduced when its entire right subtree is blank, in which case that subtree and the current tree root are removed.

This process tends to creates trees which are unbalanced to their left, which is quite far from the optimal case of balanced trees (as studied in Section 3.2). Figure 3 shows one of the worst-case outputs from this algorithm.

3.3.1. Efficiency of TreeKEM's Tree Expansion. As the tree expansion mechanism is always applied on a perfect tree (in logical representation), it can be precisely studied without the need of a simulation. And it turns out that expanding a Ratchet Tree to the right is particularly inefficient in terms of communication cost.

Indeed, the perfect tree that needs to be expanded is depth-balanced and therefore yields an optimal communication cost. Keeping that balance implies adding any additional leaf at a depth whose difference from that of the other leaves is less than or equal to one. Intuitively, this means adding the leaf as close as possible to the bottom of the tree.

However, TreeKEM's mechanism does precisely the opposite. Adding a new user in a new right subtree attached to the root, whatever this user's location in that subtree, corresponds in logical representation to adding a leaf just below the root, at depth $d = 1$. As the previous leaves are all at depth $d = \log_2(n)$ (with n the original number of users), adding a user this way induces a communication overhead of $\Delta\kappa_{expand}^{tk} = (\lambda+1)(n+1)$. This value must be compared to the optimal overhead – when the expanded tree remains depth-balanced – of $\Delta\kappa_{expand}^{\text{opt}} = (\lambda+1)(\log_2(n)+2)$.

3.3.2. Efficiency of TreeKEM's User Add. Contrary to the tree expansion process, it is not possible to generically evaluate the efficiency of a user add operation in TreeKEM. Indeed, the associated communication cost strongly depends on the tree structure before the user is added, and therefore, on the previous changes undergone by the user group. The efficiency analysis of that process is consequently analyzed experimentally in the remainder of this paper.

Despite its suboptimal communication cost, the tree evolution mechanism adopted by TreeKEM presents the advantage that the choice of a new user's location is deterministic and needs only few computations. Therefore, the committer that adds the new user has no need to broadcast that location to the entire group. Instead, any member – that has a full view of the Ratchet Tree – locally updates its own copy of that tree.

3.4. Two Improved Tree Evolution Algorithms

We present here two algorithms designed to optimize the operations of tree expansion and user add in a full binary tree such as the one used with TreeKEM, while keeping the simplicity and deterministic aspect of TreeKEM's algorithms. Both algorithms are generalizable to trees of any degree $m \geq 2$.

3.4.1. Optimal Tree Expansion. As we have previously seen, the best way to expand a perfect tree of initial height h consists in adding the additional leaves at a depth as close as possible to the one of the original leaves.

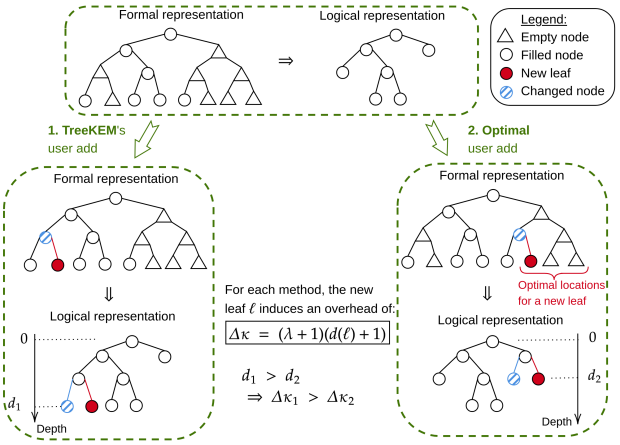


Figure 3: Communication cost of a user add in a full binary tree, with TreeKEM's left-balanced method (left) and the optimal one (right). Because this cost only depends on the depth at which the new user is inserted, left-balancing the Ratchet Tree often leads to costly tree structures.

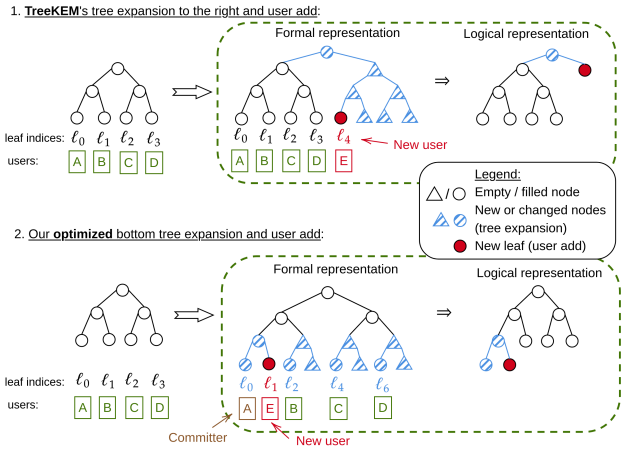


Figure 4: Ratchet Tree expansion with TreeKEM's right-expansion (top) and our optimized bottom method (bottom) that keeps the tree's depth-balance but modifies the leaf indices of the existing users.

Since the tree is perfect, the new leaves must be added at the bottom of the tree, one level below the original leaves. This ensures that the resulting tree remains depth-balanced, and that its communication cost consequently stays optimized.

In formal tree representation, this method is carried out by moving all the 2^h current leaves down one level (to depth $h+1$), replacing them at depth h by internal nodes, and by attaching a blank sibling to each of them. The bottom layer of the Ratchet Tree, at depth $h+1$, thus becomes an alternation of 2^h filled and 2^h empty leaves (cf. Figure 4) that can be used afterwards to welcome new users.

This Bottom Expansion method involves modifying the leaf indices of the existing users, which are multiplied by a factor two (in a binary tree): $\forall i \in [\![0, 2^h - 1]\!]$, $\ell'_i = 2\ell_i$.

Even if, with TreeKEM, users keep the same leaf index as long as they belong to the group, the standard clearly specifies that the correlation between a user ID and a leaf index depends on the epoch[11]. Consequently, occasionally changing the users' leaf indices does not cause any trouble, as long as all group members are aware of these changes and update their local view of the Ratchet Tree accordingly. In the case of our bottom tree expansion, no specific instruction needs to be broadcast in order to implement this index shift. Indeed, all group members receiving a commit that comprises a user add, while the Ratchet Tree is full, know that they must expand the latter by multiplying all users' indices by two.

Figure 16 (Appendix C) details the pseudocode for the Bottom Expansion, extended to the generic case of a m-ary tree.

The main drawback of our method comes from the *unmerged leaves* process, carried out by TreeKEM in order to maintain the CGKA's forward secrecy at the arrival of a new user. This process indeed blanks the whole direct path of the arriving user, so that none of these nodes contains secret elements related to previous epochs. Because all nodes above the new user are blank, the latter is, logically speaking, attached straightly to the root. Later, as the newcomer's neighbors update their own paths, that user is attached further down the tree.

Since TreeKEM's tree expansion to the right puts the new user in a separate new subtree that is already blank, no path blanking is performed on the original tree (that now constitutes the current left subtree). With our bottom tree expansion, nevertheless, we need to blank the direct path of the new user's location in the original tree, composed of h nodes, which affects in consequence the structure of half of the 2^h original leaves.

Fortunately, the effect of adding a single user can be totally offset if the protocol adds that newcomer (after expanding downward the Ratchet Tree) as the committer's sibling. Because the commit automatically updates the committer's direct path, which is identical to its sibling's, the nodes that have been blanked by the user add are filled again right after.

3.4.2. Almost-Optimal User Add. The optimal location where a new user should be added in a full binary tree is at depth $d_{min}+1$, attached to the top leaf (or one of the top leaves) at original depth d_{min} in the tree.

The reason for it is simple: as the future sibling of the new leaf moves down a level, from d_{ℓ_s} to $d'_{\ell_s} = d_{\ell_s}+1$, its parent cost is incremented by one: $\Delta c_p^{\ell_s} = 1$. Moreover, the new leaf ℓ_n induces an additional parent cost of $c_p^{\ell_n} = d_{\ell_s}+1$. The tree's parent cost therefore varies as follows: $\Delta c_p = \Delta c_p^{\ell_s} + c_p^{\ell_n} = d_{\ell_s} + 2$ and

$$\Delta\kappa_{userAdd} = \Delta c_p(\lambda+1) = (d_{\ell_s}+2)(\lambda+1) \qquad (13)$$

Consequently, mitigating the overhead associated with a user add comes to minimizing the original depth d_{ℓ_s} of the leaf to which the new leaf will be attached. In other terms, the higher we can insert the new leaf in the tree, the better it is for the tree's communication cost.

Difficulty Finding the Top Leaves of a Ratchet Tree. A naive approach to determine the top leaf/leaves in the Ratchet Tree consists in recording, in each user's state, the

[11] Indeed, due to the changes in the group membership, the protocol may assign the same leaf index to different users, at different time points.

depths of all leaves in the tree. This can be seen as an array of all the tree leaves, where each leaf index is associated with its depth. However, the computations needed to keep this array updated, after each user add and removal, and sorted by depth can become quickly costly. Indeed, adding or removing a leaf has an influence on the depth of all the leaves in the subtree rooted at that leaf's sibling. In a worst-case scenario, this number can reach 2^{h-1} in a perfect binary tree of height h.

Our Lightest Child Algorithm. Considering the difficulty to keep an up-to-date record of the leaf depths in a large tree, we have designed a simple yet effective algorithm which computes, when a user add is requested, a *close-to-optimal* insertion location for that new leaf. This algorithm is called the Lightest Child Algorithm.

The idea behind it is that the lower the weight of a tree (i.e. the fewer leaves it has), the smaller the average depth of its leaves is. Consequently, comparing sibling subtrees of the Ratchet Tree – i.e. subtrees rooted at the same depth – by considering their respective weights determines which one of them has in average the highest leaves: this one is the lightest subtree. Consequently, it is likely that the highest leaf in the tree, that we are looking for, is located precisely in that lightest subtree.

We proceed recursively by selecting the lightest subtree and comparing the weights of that subtree's subtrees, and so on, until the subtree we are working on comprises two nodes (one of which is necessarily blank, unless the Ratchet Tree is totally filled and needs an expansion). In total, in a Ratchet Tree with 2^x leaves, this operation must be carried out $x-1$ times, logarithmically in relation to the number of group members.

For efficiency considerations, instead of recursively determining the lightest subtree of each lightest subtree in the Ratchet Tree, and proceeding this way at every user add[12], we determine at the beginning of the group evolution a Weighted Ratchet Tree. This tree, depicted in Figure 5, is a copy of the Ratchet Tree in its formal representation, in which each node is associated with its weight[13].

Thanks to this Weighted Ratchet Tree, recursively comparing the lightest subtrees of the Ratchet Tree comes to selecting, from the root, the Lightest Child of each previously selected node, i.e. the node with the minimal associated weight[14]. With this method, building (once) the Weighted Ratchet Tree has a computational cost in $O(n)$ but afterwards, each use of that Weighted Ratchet Tree has a computational complexity of only $O(\log_2(n))$, as it comes to reading the direct path of a leaf in a perfect binary tree. Moreover, the creation of that Weighted Ratchet Tree is done at the beginning of the group history, when the Ratchet Tree is small; consequently the computational overhead of the Lightest Child algorithm appears quite limited in practice.

[12]Such basic approach implies to read all n users, then $\frac{n}{2}$ users of the lightest subtree, and so on during the $\log_2(n)$ stages of the process, which induces a computational cost in $O(n)$.

[13]This Weighted Ratchet Tree can also be seen as a slight modification of the Ratchet Tree, where a new weight field is added to the public state of the nodes.

[14]In case several children have an identical weight, the leftmost child is selected.

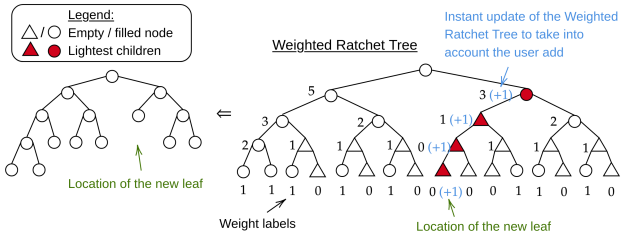


Figure 5: User add in a Ratchet Tree with our Lightest Child algorithm, relying on the associated Weighted Ratchet Tree. Determining a location for a new user requires reading only a logarithmic number of nodes in that tree.

The Weighted Ratchet Tree evolves with the group membership and must be updated at each user add or removal, by increasing or decreasing by one the weight of each node in the added/removed leaf. For efficiency considerations, the update of the Weighted Ratchet Tree during a user add is carried out at the same time as the determination of the lightest child (blue increment in Figure 5).

Regarding the storage cost induced by this algorithm, it simply consists in adding an additional field of four bytes (to store the weight variable) in the state of every node of the Ratchet Tree. This overhead is completely negligible compared to the size of these states.

Figure 17 (Appendix C) shows the pseudocode of the Lightest Child algorithm applied to a m-ary tree, including the construction of the associated Weighted Ratchet Tree.

Since the optimal location for a new leaf strongly depends, once again, on the tree structure at the time of that user add, the efficiency of our algorithm can only be assessed through experimental results. These ones, detailed below, show that the Lightest Child algorithm, paired with our Bottom Tree Expansion, gives the tree a communication cost very close to the optimal one, with an average overhead around 0.5 %.

3.5. Experimental Results

In order to assess in practice the efficiency of our optimizations, we have modeled a random evolution of a group of users and we have compared the communication costs of a full binary Ratchet Tree under this evolution pattern, both with TreeKEM's tree evolution mechanism and with our optimized one:

- With TreeKEM, as specified in Section 3.3, users are added on the leftmost possible location in the tree and the Ratchet Tree is expanded to the right.
- Our optimized method uses both our Lightest Child user add algorithm and our Bottom Tree Expansion.

The user group is initialized by consecutively adding n_{init} users to the group, which models real-life conversations where an administrator creates a group by inviting one-by-one all the desired members. The associated Ratchet Tree includes each new user, and expands if necessary, according to the tree evolution mechanism used.

After this initialization, the user group follows a random walk: at each iteration, the group undergoes, with

an equal probability, either a user add or a user removal. As in real life, the removed user is randomly selected among all (filled) leaves of the Ratchet Tree. Conversely, the location where the new user is added to that tree and the way the tree is expanded, if needed, are determined according to each tree evolution algorithm.

We underline that even if user adds and removals are not the only operations realized by a CGKA (the key updates of the existing users are indeed more frequent), these two operations are the only ones that influence the tree-dependent communication cost κ of the protocol. Once that cost is modified after the arrival or departure of a group member, it remains unchanged for every subsequent commit until the next change in the group membership (which means that the associated communication overhead impacts the cost of all that subsequent commit messages). Therefore, our simulations only take into account these two significant operations of user add and user removal.

Our simulations, an instance of which is pictured by Figure 18 in Appendix D, show that:

- the communication overhead of a Ratchet Tree evolving following TreeKEM's method mainly comes from the tree expansions;
- the user add algorithm from TreeKEM appears unable to decrease that overhead, even long after the expansion.

Consequently, the efficiency of TreeKEM is highly impacted by the number of necessary Ratchet Tree expansions during the group history. This number of expansions itself mainly depends on the initial number of users n_{init} in the group. Indeed, with a number of iterations in the random walk limited to a hundred in our simulations[15], the number n of users remains quite close to that initial value n_{init}. Therefore, when n_{init} is close to a power of two, the chances that the tree expands – which happens when $n > 2^k,\ k \in \mathbb{N}$ – appear quite high.

To take this factor into account, we have studied the average communication overheads (w.r.t. the optimal case of a depth-balanced tree), over dozens of repetitions, of Ratchet Trees following the aforementioned tree evolution methods and with an initial number of users ranging from 10 to 1,000. Table 1 depicts these results. In order to assess the specific impact of our two optimized algorithms, we have also considered separately two partially-optimized tree evolutions based on TreeKEM:

- one where only the tree expansion is improved, with our Bottom Tree Expansion;
- one where only the user add mechanism is enhanced, with our Lightest Child algorithm.

Figure 6 also illustrates the impact of the initial number of users in the tree on the communication overhead of the whole life of the user group. It confirms that the communication cost of a Ratchet Tree following TreeKEM's evolution mechanism suffers from an important overhead when the initial number of user is around a power of two and therefore induces one or several costly tree expansions.

When TreeKEM's tree expansion method is replaced by our Bottom Expansion, these communication peaks are greatly decreased, due to the more efficient tree expansion process this semi-optimized method carries out. In parallel, when one replaces TreeKEM's user add method by our Lightest Child algorithm but keeps the tree expansion to the right, the communication cost of the Ratchet Tree is lowered compared to TreeKEM, but keeps the same pattern with peaks of communication cost when the number of users is around of power of two. Finally, the fully optimized evolution mechanism – that implements both our improved algorithms – offers the most efficient communication cost, regardless of the initial number of users, thanks to the advantages of its two component algorithms.

In order to further improve the tree-dependent communication cost of a Ratchet Tree, we now wonder whether the binary tree used in TreeKEM as a Ratchet Tree is best suited for that protocol, or whether another tree degree would appear more appropriate.

4. Optimizing the Tree Degree as a Function of the Communication Ratio

The degree of a Ratchet Tree influences its shape and therefore the communication cost associated with it. Namely, for a fixed number of users, trees with a higher degree are wider (and shallower) while trees with a lower degree are thinner (and deeper). According to Equation (6), this means that schemes with a high ratio λ should be associated with a higher tree degree.

Table 3 in Appendix B details the communication ratios for the encryption schemes that are expected to be used with MLS, both in the classical and in the post-quantum frameworks. It emerges from this table that all classical ciphersuites are associated with a communication ratio $\lambda \in [0.4, 0.6]$, whereas most promising PQ ciphersuites – including the newly strandardized ML-KEM – have a higher ratio $\lambda \in [0.9, 1.0]$. The next paragraphs are devoted to determining, with theoretical bounds tightened by experimental simulations, the optimal tree degree in our parameter range $\lambda \in [0.4, 1.0]$.

TABLE 1: Tree-dependent communication overhead of a Ratchet Tree evolving with TreeKEM's tree evolution mechanism and with our improved algorithms. These values are computed in average for a range of initial number of users that is indicated in the first column.

Initial nb of users n_{init}	Communication overhead w.r.t. the optimal cost (%)			
	TreeKEM	Optim. tree expansion	Optim. user add	Fully optim.
10-20	5.99	4.58	1.64	0.61
20-50	5.03	3.49	1.28	0.40
50-100	4.12	1.73	1.34	0.19
500-550	6.94	0.35	5.72	0.07
1,000-1,050	4.99	0.11	4.47	0.02

[15]This number of iterations, that may seem low at first sight, represents a hundred changes in the group membership, which corresponds in real use-cases to long-life or very active conversation groups.

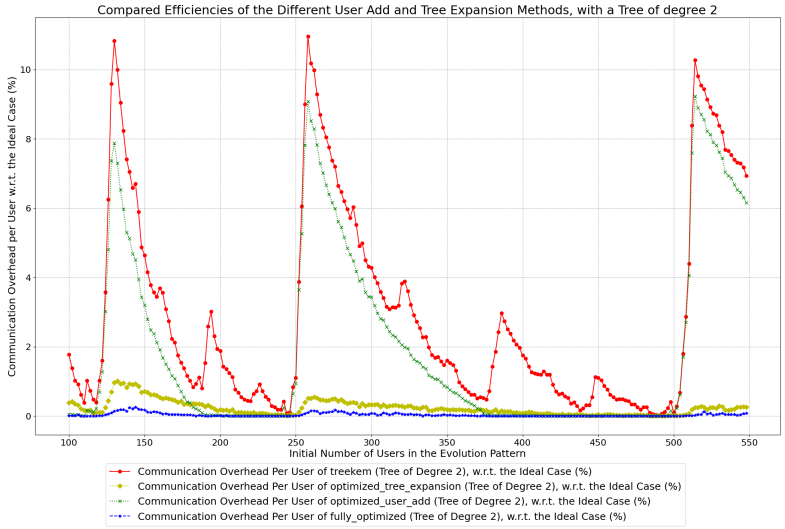


Figure 6: Compared communication costs of randomly evolving Ratchet Trees with initial numbers of users ranging from 100 to 550, with TreeKEM's tree evolution mechanism and our optimized algorithms.

4.1. The Optimal Tree in a Family

Let $\kappa_\lambda(T)$ be the tree-dependent cost function from Equation (6) for a given communication ratio λ. For any tree T, we also define $\nu(T)$ as the point $(c_p(T), c_s(T))$ in $\mathbb{R}^2$.

We extend these notations to sets of trees: for any set $\mathcal{S}$ of trees with n leaves, let $\nu(\mathcal{S}) = \{\nu(T),\ T \in \mathcal{S}\} \subset \mathbb{R}^2$, and we define the *optimal cost function* as:

$$\kappa_\lambda(\mathcal{S}) = \inf\{\kappa_\lambda(T),\ T \in \mathcal{S}\} \tag{14}$$

Proposition 1. The piecewise affine function $\kappa_\lambda(\mathcal{S})$ is dual to the lower-left boundary $\mathcal{B}$ of the convex hull H of the set $\nu(\mathcal{S})$: slopes of $\mathcal{B}$ correspond to vertices of $\kappa_\lambda(\mathcal{S})$, while vertices of $\mathcal{B}$ correspond to slopes of $\kappa_\lambda(\mathcal{S})$.

Nota: Since each function κ_λ is an affine function of λ, $\kappa_\lambda(\mathcal{S})$ may also be seen as the opposite of the Fenchel transform [28, 3.3] of $\mathcal{B}$.

Proof 2. Let $\nu(\mathcal{S})^+$ be the Minkowski sum [29] of $\nu(\mathcal{S})$ and the upper-right quadrant of $\mathbb{R}^2$. The lower-left boundary of H is also the boundary $\mathcal{B}$ of the convex hull H^+ of $\nu(\mathcal{S})^+$, as depicted in Figure 7. All elements of $\mathcal{S}$ that have – for a given value λ – the same communication cost $\kappa(T) = c_p(T)\lambda + c_s(T)$, lie on an affine line with slope $-\lambda$. Therefore, the trees that are *minimal* for this value λ lie on $\mathcal{B}$.

The boundary $\mathcal{B}$ is a polygonal line whose vertices correspond to trees that are optimal over a range of values of λ: namely, let $\nu(T_i)$ be the vertex of $\mathcal{B}$ at the intersection of the segments with slopes $-\lambda_i$ and $-\lambda_{i+1}$, then for any $\lambda \in [\lambda_i, \lambda_{i+1}]$ and any $T' \in \mathcal{S}$, $\kappa_{T'}(\lambda) \geq \kappa_{T_i}(\lambda)$, as displayed on Figure 7. The slopes of the segments of $\mathcal{B}$ thus correspond to "critical" values of λ, marking the boundary between two different optimal trees. □

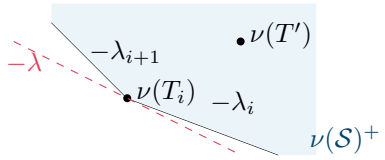


Figure 7: Optimal trees in a set $\mathcal{S}$, viewed as vertices of the lower-left part of the convex hull of $\nu(\mathcal{S})^+$.

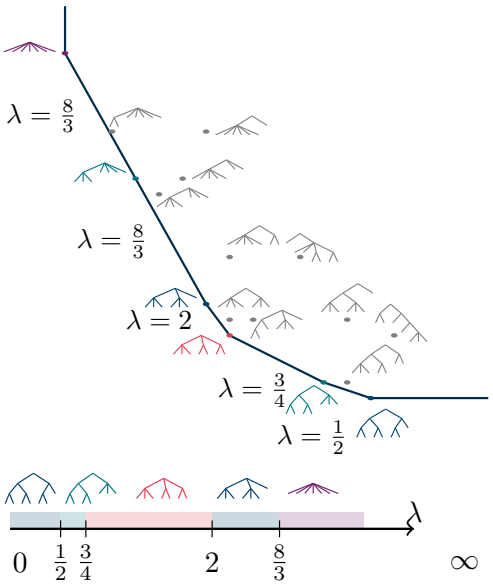


Figure 8: Top: the set of optimal trees with 7 leaves, as a function of the plaintext ratio λ. Some non-optimal trees are also displayed (in gray). Bottom: the associated optimal cost κ_λ as a function of λ, with the individual optimality domains highlighted.

4.1.1. Finally, a Map of Every Tree. [16] As an example, Figure 8 displays the optimal trees among the 90 irreducible, non-planar trees with 7 leaves. The computation confirms that the binary tree is optimal for small plaintext ratios ($\lambda \leq \frac{1}{2}$) while the wide comb-like tree is optimal

[16] See https://xkcd.com/2977.

for $\lambda \geq \frac{8}{3}$. When λ is exactly equal to $\frac{8}{3}$, there are actually three co-optimal trees, with (c_p, c_s) costs of $(7, 42)$, $(10, 34)$ and $(13, 26)$. Since the second of those trees is optimal neither for $\lambda < \frac{8}{3}$ nor for $\lambda > \frac{8}{3}$, in practice it may safely be ignored.

In the general case, the complexity of computing the function $\kappa_\lambda(\mathcal{S})$ over a finite set $\mathcal{S}$ of N trees is that of computing the convex hull of a planar set of N points, which is $O(N \log(N))$ [30].

On the other hand, the cardinality N_n of the set of all irreducible trees with exactly n leaves is given by the OEIS sequence A000669 [31] [25, I.45], which grows exponentially: $N_n = \Omega(\rho^n)$ for some real number $\rho \geq 3.5$ [25, VII.5]. This restricts the enumeration of all trees to small values of n. We did a complete survey of all trees up to $n = 25$; the optimal trees are listed in artifact best_trees.toml[17].

While an extensive study is out of computational reach, we can focus on specific families of trees that seem to be of interest in practical ranges of λ intuited by our exhaustive study of small trees stated above. As a first step, we provide below an analysis on full trees.

4.1.2. Optimal Degree of a Full Tree. As we have previously proved that a m-ary tree minimizes its communication cost when it is depth-balanced, determining the optimal degree of full trees comes to analyzing the set of depth-balanced full trees of various degrees. When the number of leaves in that trees is fixed, that comparison is easily done with the above convex hull method[18]. This basic study leads to Figure 9, which shows that:

- The exact number of leaves in the tree has a significant influence on the optimal degree: namely, perfect trees (depth-balanced trees whose number of leaves is a power of m) are optimal for large intervals of the parameter λ. This factor tends to be influential on small trees but becomes asymptotically negligible.
- The optimal degree of a full tree increases along with the parameter λ.

However, when its degree exceeds two, a CGKA's Ratchet Tree never stays full. Indeed, a m-ary tree can be full only when the number of leaves is $n = m + k(m-1),\ k \in \mathbb{N}$ and when these leaves are equally distributed within that tree. As we do not control when and where users leave the group (i.e. when and where leaves are removed from the tree), our analysis on the optimal tree degree must extend to the case of the numerous non-full trees.

4.2. Degree Bounds for Practical Values of λ

4.2.1. Tree Collapsing. We now introduce *weighted trees* as a tool allowing us to study the inner structure of a tree while abstracting away its branches.

Definition 5. A *weighted* (non-plane) tree is either a leaf ℓ, together with a real number w called its *weight*, or an unordered set $\{B_1, \ldots, B_m\}$ of weighted trees.

[17]This artifact will be made public at time of publication of this paper, and can be provided to the reviewers on request.

[18]We can also use Equation (7) in Section 3.2.1, that gives the communication cost of a depth-balanced full tree of degree m, to compare the area of optimality of trees with consecutive degrees.

Ordinary trees correspond to trees where each leaf has weight one. The definition of tree weight from Section 2.2.2 naturally extends to weighted trees: the weight $w(T)$ of a weighted tree T is the sum of the weight of all of its branches (equivalently, of all its leaves).

We also define the cost functions functions c_p and c_s for weighted trees in the same way as in Section 3.1.1, taking care to use the extended weight function $w(B_i)$ for all branches B_i.

Theorem 2 (Tree Collapsing). Let T be a tree (weighted or not), B an inner branch of T, and T' be the weighted tree obtained by replacing B by a single leaf with weight $w(B)$. Then:

$$c_p(T) = c_p(T') + c_p(B) \tag{15}$$
$$c_s(T) = c_s(T') + c_s(B) \tag{16}$$

Proof. We proceed by induction on the collapsed tree T'. If T' is a leaf then $c_p(T') = 0$ while $T = B$: the result is then obvious.

Let now $T = \{T_1, \ldots, T_m\}$; assume without loss of generality that B is a sub-branch of T_1 and let T_1' be the corresponding collapsed tree. Then $T' = \{T_1', T_2, \ldots, T_m\}$ and in particular $w(T') = w(T)$. By definition, $c_p(T) = w(T) + \sum_i c_p(T_i)$; therefore $c_p(T) - c_p(T') = c_p(T_1) - c_p(T_1')$. By the induction hypothesis, this is equal to $c_p(B)$.

The same result holds for $c_s(T) - c_s(T')$. □

Theorem 2 allows replacing the study of a tree T, (having an arbitrarily large number of leaves) by that of a small collapsed tree T', where each leaf represents a whole branch of the original tree. In particular, re-arranging the branches of the tree T has the same impact on the costs c_p and c_s as the corresponding re-arrangement of the leaves of the collapsed tree T'.

4.2.2. Trees with Three Leaves. We now describe the configuration space for the optimal trees with either two or three branches at their root, depending on the ratio λ as well as on the relative sizes of the branches. For this, we study the weighted trees with three leaves. There are only two such trees: the binary tree $T_2 = \{\ell_1, \{\ell_2, \ell_3\}\}$ and the ternary tree $T_3 = \{\ell_1, \ell_2, \ell_3\}$. Let w_1, w_2, w_3 be the weights of the leaves; without loss of generality we may assume $w_1 + w_2 + w_3 = 1$. The costs associated to these trees are straightforward:

$$c_p(T_2) = c_s(T_2) = w_1 + 2(w_2 + w_3) \tag{17}$$
$$c_p(T_3) = w_1 + w_2 + w_3 \tag{18}$$
$$c_s(T_3) = 2(w_1 + w_2 + w_3) \tag{19}$$

These costs compare in the following way:

$$\kappa_{T_3}(\lambda) \leq \kappa_{T_2}(\lambda) \ \Leftrightarrow\ \lambda \geq \frac{w_1}{1 - w_1} \tag{20}$$

On the other hand, both trees T_3 and T_2 have symmetries. Namely, using the symmetry of the tree T_3, one may assume that w_1 is the largest of the three weights, and in particular that $w_1 \geq \frac{1}{3}$. In the same way, for any tree with two branches and at least four leaves – represented by T_2 – we can select which branch is collapsed into the weighted leaf ℓ_1 and which branches into the leaves $\{\ell_2, \ell_3\}$. Since the cost of such a tree increases with w_1, the optimal tree

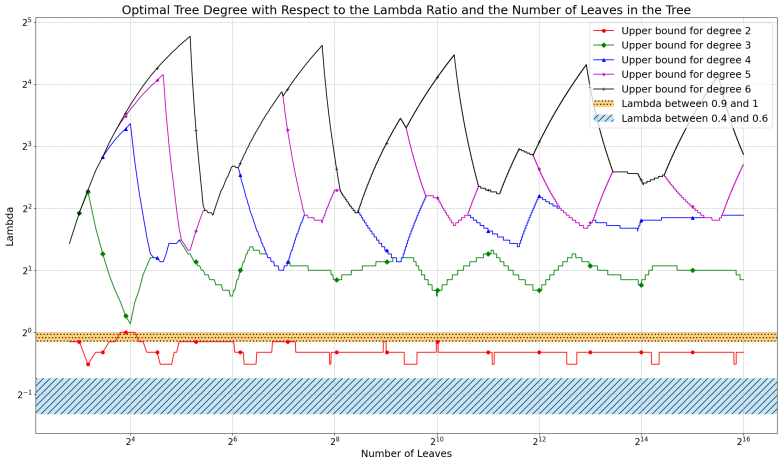


Figure 9: Areas of optimal tree degree for a *full* tree, according to the values of the communication ratio λ and the number of users in the tree (in a logarithmic scale).

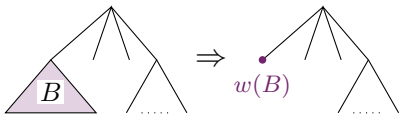


Figure 10: The weighted tree resulting from a collapse.

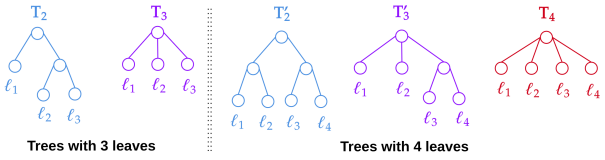


Figure 11: Small trees with three or four branches.

is reached when the ℓ_1 branch is the lightest branch from the root of the tree, which implies that $w_1 \leq \frac{1}{2}$. Putting all these results together, we can represent the set of optimal weighted trees with three leaves as in Figure 12a.

In particular, we deduce the following:

Proposition 2.

1) For $\lambda < \frac{1}{2}$, the optimal trees are binary.
2) For $\lambda > 1$ and $n \geq 3$, the optimal trees with n leaves are *at least* ternary.

For $\lambda \in [\frac{1}{2}, 1]$, the optimal tree configurations actually have a mixture of binary and ternary nodes; this will be further studied in Section 4.3.

4.2.3. Trees with Four Leaves. Of the five trees with four leaves, the only one which cannot be decomposed using the previously defined families T_2 and T_3, is the flat tree $T_4 = \{\ell_1, \ell_2, \ell_3, \ell_4\}$.

We can compare this tree with $T'_3 = \{\ell_1, \ell_2, \{\ell_3, \ell_4\}\}$ and with $T'_2 = \{\{\ell_1, \ell_2\}, \{\ell_3, \ell_4\}\}$ (cf. Figure 11). Using the same methods as previously, we find that:

1) $\kappa_{T_4}(\lambda) \leq \kappa_{T'_3}(\lambda)$ for $\lambda \geq \frac{w_1+w_2}{w_3+w_4}$;
2) $\kappa_{T_4}(\lambda) \leq \kappa_{T'_2}(\lambda)$ for $\lambda \geq 1$.

Moreover, by using the symmetries for T_4 or T'_2, one may always ensure that $w_3 + w_4 \geq \frac{1}{2}$ in both cases. Furthermore, let T be a large enough tree with a ternary root. We can always collapse that tree in a way such that the leaves ℓ_3 and ℓ_4 are the two heaviest ones; this ensures that $w_3 + w_4 \geq \frac{1}{3}$. We thus obtain the configuration space shown in Figure 12b and the following conclusions:

Proposition 3.

1) For $\lambda < 1$, the optimal trees are at most ternary.
2) For $\lambda > 2$ and $n \geq 4$, the optimal trees with n leaves are *at least* quaternary.

4.2.4. An Upper Bound on the Optimal Degree for an Arbitrary Communication Ratio. The previous section completely defines the domains where binary and ternary trees can be optimal. For general values of the communication ratio λ, we now give an upper bound on the theoretically optimal degree m.

Finding a lower bound seems to be a harder problem. However, in practice the dimensioning parameter for implementation is the maximal tree degree. We therefore expect that the upper bound will be the most interesting one.

For any integer b and normalized weights $w_1 + \cdots + w_b = 1$, we compare the two small weighted trees:

$$T_b = \{\ell_1, \ldots, \ell_b\} \quad \text{and } T'_{b-1} = \{\{\ell_1, \ell_2\}, \ell_3, \ldots, \ell_b\}. \tag{21}$$

T_b and T'_{b-1} both have b leaves but different degrees. Let $x = w_1 + w_2$ be the joint weight of ℓ_1 and ℓ_2. We have:

$$\begin{aligned} c_p(T'_{b-1}) &= c_p(T_b) + x \\ c_s(T'_{b-1}) &= c_s(T_b) - (1 - x) \\ \kappa_{T'_{b-1}}(\lambda) < \kappa_{T_b}(\lambda) \quad &\Leftrightarrow \quad x(\lambda + 1) < 1 \end{aligned} \tag{22}$$

Using the symmetry between the branches, we may always choose ℓ_1, ℓ_2 as the two heaviest leaves, so that $x \geq \frac{2}{b}$. Therefore:

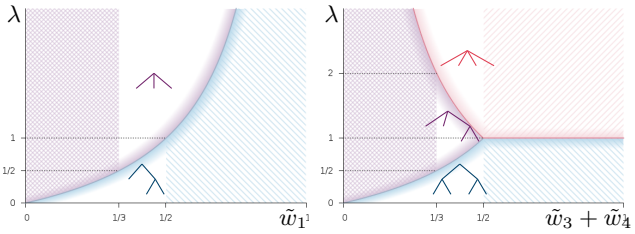
$$\kappa_{T'_{b-1}}(\lambda) < \kappa_{T_b}(\lambda) \quad \Leftrightarrow b > 2(\lambda + 1) \tag{23}$$

This means that it remains interesting to decrease the degree of any node from a tree, from b to $b - 1$, as long as $b > 2(\lambda + 1)$. Consequently:

Proposition 4. For any communication ratio λ, the optimal tree degree m^{opt} is upper bounded by $m^{\text{opt}} \leq 2(\lambda + 1)$.

4.3. Experimental Comparison of Non-Full Trees

The study above shows that when $\lambda \in [0.5, 1]$, the optimal tree degree is two, three or even four (for the upper bound $\lambda = 1$). However, it does not permit to theoretically



(a) Trees with three leaves (b) Trees with four leaves

Figure 12: Optimal tree configurations with a small number of leaves. The crossed-out areas mark the configurations which can be eliminated by symmetries.

select one of these degrees because the optimal value not only depends on λ and on the number of leaves in the tree, but also on that tree's structure, i.e. on the relative disposition of the nodes, that cannot be considered in a general case.

Consequently, we have carried out simulations using the same program as in Section 3.5, during which we have compared the average communication cost per user[19], both with TreeKEM's methodology and with our fully-optimized method, of:

- ternary and quaternary trees, for $\lambda \geq 1$;
- binary and ternary trees, for $\lambda \in [0.5, 1]$.

4.3.1. Ternary vs Quaternary Trees. The simulation, whose results are depicted in Figure 19 in Appendix D, shows that the ternary tree remains more efficient than the quaternary one as long as $\lambda^{3-4} \in [1.45, 1.70]$ – depending on the tree evolution mechanism used –, which corresponds to the use case of most classical or post-quantum HPKE ciphersuites.

4.3.2. Binary vs Ternary Trees. Similarly, Figure 13 shows a limit ratio $\lambda^{2-3} \in [0.70, 0.75]$ under which a binary tree has a better communication cost than a non-full ternary one[20]. This limit appears quite similar to the one between binary and *full* ternary trees (cf. Figure 20 in Appendix D), which also varies in the interval $[0.7, 0.8]$, with an asymptotic limit (regarding the number of leaves in the tree) of 0.73.

Figure 14 sums up the tree degree adapted to various classical and post-quantum ciphersuites used for HPKE encryption. It underlines that in the post-quantum setting, and especially with the standardization of ML-KEM, the Ratchet Trees used within MLS should be ternary.

Finally, Table 2 depicts the communication overhead of the current binary tree used by TreeKEM, compared to a ternary tree evolving with that same method or according to our optimized one, when $\lambda = 1$ which corresponds to the post-quantum framework. The difference between TreeKEM's binary tree and our optimized ternary tree is about 10%, which is particularly beneficial in the post-quantum setting where the amount of exchanged data is large and therefore where this percentage appears non-negligible.

[19]These average results are computed from random walks whose initial numbers of users vary from 10 to 1,000, as in Section 3.5.

[20]As only irreducible trees are considered here, binary trees are always full.

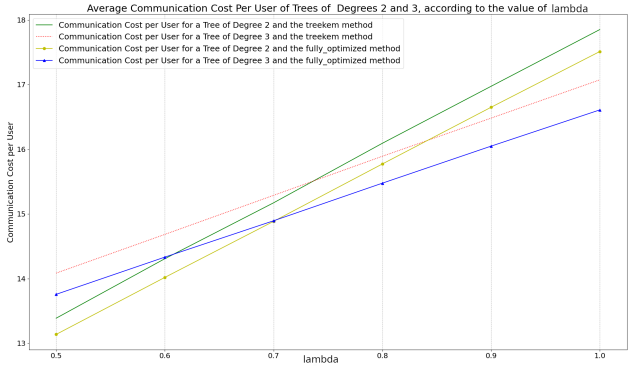


Figure 13: Compared communication costs per user of binary and ternary trees evolving with TreeKEM's and our optimized methods. It shows an optimality bound between these two degrees at $\lambda \in [0.70, 0.75]$.

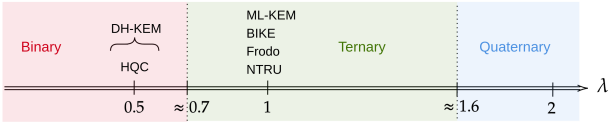


Figure 14: Optimal degree of a CGKA's Ratchet Tree, according to the HPKE ciphersuite used.

5. Conclusion

We have shown in this paper the sub-optimality of the Ratchet Tree used in TreeKEM in terms of communication cost. The binary tree currently used in that CGKA is adapted to the classical HPKE ciphersuites but not to the PQ ones, that yet tend to be included in all recent encryption protocols in the framework of PQ hybridization. In that context, the use of a ternary tree appears more efficient, while being as easy to implement as its binary counterpart.

Moreover, we have proposed two algorithms that add new users and expand the Ratchet Tree more efficiently than the current processes of TreeKEM. These algorithms induce limited computational and memory overhead and, on the other side, gain around 5% of bandwidth.

When combining our algorithmic enhancements and the use of a ternary tree, we lower the communication cost of TreeKEM of around 10%, in the context of post-quantum encryption where every byte of bandwidth gained is appreciable.

TABLE 2: Tree-dependent communication overhead of a binary Ratchet Tree following TreeKEM's evolution mechanism, compared to ternary trees evolving respectively with TreeKEM's and our improved evolution algorithms, for a ratio $\lambda = 1$ expected in the post-quantum framework.

Initial nb of users n_{init}	Communication overhead (%) of TreeKEM binary, w.r.t. TreeKEM ternary	Fully optim. ternary
10-20	6.45	10.69
20-50	5.68	9.97
50-100	5.00	9.81
500-550	9.73	12.38
1,000-1,050	7.74	9.74

References

[1] "WhatsApp Encryption Overview," WhatsApp Inc., Technical White Paper, January 2023. [Online]. Available: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf

[2] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol," RFC 9420, Jul. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9420

[3] J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis, "Security analysis and improvements for the IETF MLS standard for group messaging," in Advances in Cryptology – CRYPTO 2020, Part I, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12170. Springer, Cham, Aug. 2020, pp. 248–277.

[4] E. J. Harder and D. M. Wallner, "Key Management for Multicast: Issues and Architectures," RFC 2627, Jun. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2627

[5] C. K. Wong, M. G. Gouda, and S. S. Lam, "Secure group communications using key graphs," in Proceedings of ACM SIGCOMM, Vancouver, BC, Canada, Aug. 31 – Sep. 4, 1998, pp. 68–79.

[6] D. M. Balenson, D. McGrew, and D. A. T. Sherman, "Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization," Internet Engineering Task Force, Internet-Draft draft-balenson-groupkeymgmt-oft-00, Mar. 1999, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-balenson-groupkeymgmt-oft/00/

[7] A. Perrig, D. X. Song, and J. D. Tygar, "ELK, A new protocol for efficient large-group key distribution," in 2001 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, May 2001, pp. 247–262.

[8] A. Perrig, "Efficient collaborative key management protocols for secure autonomous group communication," in Proceedings of the International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC), Jul. 1999, pp. 192–202. [Online]. Available: /publications/papers/secmcast.pdf

[9] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in ACM CCS 2000: 7th Conference on Computer and Communications Security, D. Gritzalis, S. Jajodia, and P. Samarati, Eds. ACM Press, Nov. 2000, pp. 235–244.

[10] ——, "Tree-based group key agreement," ACM Trans. Inf. Syst. Secur., vol. 7, no. 1, pp. 60–96, feb 2004. [Online]. Available: https://doi.org/10.1145/984334.984337

[11] S. Zheng, D. Manz, and J. Alves-Foss, "A communication–computation efficient group key algorithm for large and dynamic groups," Computer Networks, vol. 51, no. 1, pp. 69–93, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128606000922

[12] Y. Desmedt, T. Lange, and M. Burmester, "Scalable authenticated tree based group key exchange for ad-hoc groups," in FC 2007: 11th International Conference on Financial Cryptography and Data Security, ser. Lecture Notes in Computer Science, S. Dietrich and R. Dhamija, Eds., vol. 4886. Springer, Berlin, Heidelberg, Feb. 2007, pp. 104–118.

[13] T. Brecher, E. Bresson, and M. Manulis, "Fully robust tree-Diffie-Hellman group key exchange," in CANS 09: 8th International Conference on Cryptology and Network Security, ser. Lecture Notes in Computer Science, J. A. Garay, A. Miyaji, and A. Otsuka, Eds., vol. 5888. Springer, Berlin, Heidelberg, Dec. 2009, pp. 478–497.

[14] 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 ACM CCS 2018: 25th Conference on Computer and Communications Security, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM Press, Oct. 2018, pp. 1802–1819.

[15] R. Barnes, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert, "The Messaging Layer Security (MLS) Protocol," Internet Engineering Task Force, Internet-Draft draft-ietf-mls-protocol-00, Aug. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/00/

[16] K. Bhargavan, R. Barnes, and E. Rescorla, "TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS)," Inria Paris, Research Report, May 2018. [Online]. Available: https://inria.hal.science/hal-02425247

[17] R. Barnes, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert, "The Messaging Layer Security (MLS) Protocol," Internet Engineering Task Force, Internet-Draft draft-ietf-mls-protocol-02, Oct. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-mls-protocol/02/

[18] S. Lee, Y. Kim, K. Kim, and D.-H. Ryu, "An efficient tree-based group key agreement using bilinear map," in ACNS 03: 1st International Conference on Applied Cryptography and Network Security, ser. Lecture Notes in Computer Science, J. Zhou, M. Yung, and Y. Han, Eds., vol. 2846. Springer, Berlin, Heidelberg, Oct. 2003, pp. 357–371.

[19] H.-H. Kim and S.-J. Kim, "A ternary tree-based authenticated group key agreement for dynamic peer group," 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:60089781

[20] A. Joux, "A one round protocol for tripartite Diffie–Hellman," Journal of Cryptology, vol. 17, no. 4, pp. 263–276, Sep. 2004.

[21] S. Tripathi and G. P. Biswas, "Design of efficient ternary-tree based group key agreement protocol for dynamic groups," ser. COMSNETS'09. IEEE Press, 2009, pp. 30–35.

[22] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman key distribution extended to group communication," in ACM CCS 96: 3rd Conference on Computer and Communications Security, L. Gong and J. Stern, Eds. ACM Press, Mar. 1996, pp. 31–37.

[23] A. Kumar and S. Tripathi, "Ternary tree based group key agreement protocol over elliptic curve for dynamic group," International Journal of Computer Applications, vol. 86, 12 2013.

[24] K. Hashimoto, S. Katsumata, E. Postlethwaite, T. Prest, and B. Westerbaan, "A concrete treatment of efficient continuous group key agreement via multi-recipient PKEs," in ACM CCS 2021: 28th Conference on Computer and Communications Security, G. Vigna and E. Shi, Eds. ACM Press, Nov. 2021, pp. 1441–1462.

[25] P. Flajolet and R. Sedgewick, Analytic combinatorics. cambridge University press, 2009.

[26] F. Harary and E. M. Palmer, Graphical enumeration. Elsevier, 1973.

[27] R. Barnes, K. Bhargavan, B. Lipp, and C. A. Wood, "Hybrid public key encryption," RFC 9180, Feb. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9180

[28] J. M. Borwein, A. S. Lewis et al., Convex Analysis and Nonlinear Optimization. Canadian Mathematical Society/Société mathématique du Canada, 2006.

[29] G. Ewald, Combinatorial Convexity and Algebraic Geometry. Springer Graduate Texts in Mathematics 168, 1996.

[30] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," Info. Proc. Lett., vol. 1, pp. 132–133, 1972.

[31] N. J. A. Sloane and J. Riordan, "Sequence A000669 in the Online Encyclopedia of Integer Sequences (n.d.)," https://oeis.org/A000669, accessed 2024-04-26.

[32] C. Aguilar-Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, G. Zémor, J. Bos, A. Dion, J. Lacan, J.-M. Robert, and P. Veron, "HQC," National Institute of Standards and Technology, Tech. Rep., 2022, available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[33] P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, D. Stehlé, and J. Ding, "CRYSTALS-KYBER," National Institute of Standards and Technology, Tech. Rep., 2022, available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[34] N. Aragon, P. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Guneysu, C. Aguilar-Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, G. Zémor, V. Vasseur, S. Ghosh, and J. Richter-Brokmann, "BIKE," National Institute of Standards and Technology, Tech. Rep., 2022, available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[35] M. Naehrig, E. Alkim, J. Bos, L. Ducas, K. Easterbrook, B. LaMacchia, P. Longa, I. Mironov, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila, "FrodoKEM," National Institute of Standards and Technology, Tech. Rep., 2020, available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[36] Z. Zhang, C. Chen, J. Hoffstein, W. Whyte, J. M. Schanck, A. Hulsing, J. Rijneveld, P. Schwabe, and O. Danba, "NTRUEncrypt," National Institute of Standards and Technology, Tech. Rep., 2019, available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-2-submissions.

[37] M. R. Albrecht, D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. Lange, V. Maram, I. von Maurich, R. Misoczki, R. Niederhagen, K. G. Paterson, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, C. J. Tjhai, M. Tomlinson, and W. Wang, "Classic McEliece," National Institute of Standards and Technology, Tech. Rep., 2022, available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions.

[38] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé, "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM," Cryptology ePrint Archive, Report 2017/634, 2017. [Online]. Available: https://eprint.iacr.org/2017/634

[39] National Institute of Standards and Technology, "Module-lattice-based key-encapsulation mechanism standard," FIPS 203 ipd, aug 2023. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.203.ipd

Appendix A. Details on the TreeKEM Protocol

We give hereunder a brief description of how TreeKEM – as standardized in RFC 9420 [2] – works as a Continuous Group Key Agreement (CGKA) protocol.

A.1. Continuous Group Key Agreement

A CGKA is a sub-protocol of a Secure Group Messaging protocol, that aims to securely generate a group key which is common to all group members and evolves over time, periodically and following changes in the group membership.

Definition 6 (Propose & Commit CGKA (adapted from [3])).
A CGKA with the Propose & Commit Paradigm is a tuple of the following algorithms:

- **Initialization**: user u_i creates its initial state γ_i: $\boxed{\gamma_i \leftarrow \mathbf{init}(u_i)}$
- **Group Creation**: user u_i, with state γ_i, creates a new group that must include users from the list $G = (u_i)_{i \in [\![1,n]\!]}$. A message welcome W is sent to all members from G, with the information necessary to join the group: $\boxed{(\gamma_i', W) := \mathbf{create\text{-}group}(\gamma_i, G)}$
- **Propose**: user u_i proposes a change to the group's state through an action $a \in \mathbb{A}$, with $\mathbb{A} \supseteq \{\text{Add}, \text{Remove}, \text{Update}\}$ the set of actions authorized by the CGKA. In particular:
  - **add**(u_j): u_i proposes to add user u_j to the group;
  - **remove**(u_j): u_i proposes to remove u_j from the group;
  - **update**: u_i updates its own encryption keying material (the one of its leaf) and generates an updated state γ_i'.

  User u_i then broadcasts a Proposal message P to the entire group: $\boxed{(\gamma_i', P) \leftarrow \mathbf{propose}(\gamma_i, a\ [, u_j])}$
- **Commit**: when receiving a set of p proposal messages $\vec{P} = \{P_i \in \mathbb{A}\}_{i \in [\![1,p]\!]}$, user u_i validates them and updates its own encryption keying material and the one of its direct path, generating a new group key k. It then updates its state into γ_i' to take into account that changes, and broadcasts a Commit message Com as well as (potentially) a Welcome message for the new group members: $\boxed{(\gamma_i', k, Com\ [, W]) \leftarrow \mathbf{commit}(\gamma_i, \vec{P})}$
- **Process**: user u_i processes a Commit message Com or a Welcome Message W it has received from a committer, updates accordingly its own state and computes the new group key k resulting from these changes: $\boxed{(\gamma_i', k) := \mathbf{process}(\gamma_i, m \in \{Com, W\})}$

  Nota: The state γ_i of any user u_i is composed of a public part γ_i^{pub}, known by every group member (which includes a complete view of the Ratchet Tree, cf. Section 2.3.1), and a private part γ_i^{priv}, that this user keeps secret and that is needed to recover the group keys generated by other members.

A CGKA must fulfill the following properties, stated informally below. These properties are captured by the CGKA security game (in a game-based security model) such as the one described in figure 1 of [3][21].

- **Correctness**: every user in the group must compute the same group key.
- **Privacy**: a group key is indistinguishable from a random value for an adversary who has access to the transcript of handshake messages exchanged within the group until the generation of that group key.
- **Forward Secrecy**: the corruption of any user at some epoch does not leak any secret element (neither the group key nor the secret seeds and keys) from previous epochs.
- **Post-Compromise Security**: following the corruption of any user, the tree's secret elements become secret again after the update of all the corrupted users (provided that the adversary stays passive during these updates).

A.1.1. Node's state. Each node of this Ratchet Tree, except for the root, is associated with a local state with public and private components.

- The public state γ comprises, among other elements
  - for an internal node v: its public encryption key pk_v;
  - for a user (leaf) u_i: its public encryption and signature keys pk_i and spk_i, with the related credentials. It also includes the signature, under the user's private signature key, of the other fields of that public state.
- The private state contains:
  - the group key k and all the group secrets derived from it;

[21] This security model is not recalled in this paper, as it is outside the scope of our study.

– the private encryption keys of that node and of its filtered direct path, as well as the temporary secret elements (leaf secret, path secrets) associated with that keys.

A.2. Updates with TreeKEM

The update of the encryption keying material is implemented differently in TreeKEM whether it belongs to a user (i.e. a leaf) or an internal node.

Indeed, as stated in Definition 6, all tree operations are performed in two rounds with the Propose & Commit paradigm from TreeKEM:

- a first one where any user is free to submit *proposals* (adding new users, removing current group members, updating its own keying material...);
- a second one where the valid proposals are grouped together and implemented within a *commit* by a single user, called committer.

A.2.1. Update of the committer's filtered direct path. During a commit process, as shown by Figure 15, the committer randomly draws a secret seed called leaf secret; this one is derived, with a key derivation function, into a node secret that serves as a seed to deterministically generate a fresh encryption key-pair.

In parallel, the leaf secret is derived into another secret ps_{v_1}, called a path secret, that is associated with this leaf's parent v_1. This path secret ps_{v_1} is itself derived into a node secret to deterministically generate an encryption key-pair for the benefit of that leaf's parent v_1. It is then derived once again into a new path secret ps_{v_2}, related to another node v_2, higher in the leaf's filtered direct path, and so on, up to the tree root.

The group key k is then computed by deriving the root's path secret ps_{root}.

A.3. Tree evolution and epochs

The evolution of the group over time is represented by the notion of epoch. Each epoch corresponds to a given state of the user group, with a certain group key. Each time this group state is modified by a commit, the group key evolves and the epoch is incremented of one unit.

Appendix B. Values of the Communication Ratio with HPKE Ciphersuites

Let us consider the values of the communication ratio λ corresponding to the encryption schemes that are expected to be used with MLS. The standard for this protocol [2] specifies that the encryption is performed according to the Hybrid Public Key Encryption paradigm (HPKE, cf. [27]), using a Key Encapsulation Mechanism (KEM) to exchange a symmetric key and an Authenticated Encryption with Associated Data (AEAD) scheme to symmetrically encrypt the plaintext. The AEAD schemes recommended by that standard are AES-GCM and Chacha20-Poly1305, that both yield a 16-byte-long authentication tag.

TABLE 3: Communication ratio λ and optimal tree degree for the main classical and post-quantum HPKE ciphersuites. Sizes are given in bytes. Last column gives the optimal tree for the considered HPKE.

KEM Type		$\|ps\|$	$\|pk\|$	$\|ct\|$	λ	**Opt. Tree**
Classical HPKE Ciphersuites						
DH-KEM [27]	X25519	32	32	80	0.4	Bin
	P256	32	65	113	0.6	
	P384	48	97	161	0.6	
	X448	64	56	136	0.4	
	P521	64	133	213	0.6	
Post-Quantum HPKE Ciphersuites						
HQC [32]	128	32	2,249	4,561	0.5	Bin
	192	32	4,522	9,106	0.5	
	256	32	7,245	14,549	0.5	
ML-KEM [33]	512	32	800	848	1.0	Tern
	768	32	1,184	1,168	1.0	
	1024	32	1,568	1,648	1.0	
BIKE [34]	Level 1	32	1,541	1,653	1.0	Tern
	Level 3	32	3,083	3,195	1.0	
	Level 5	32	5,122	5,234	1.0	
Frodo [35]	640	32	9,616	9,800	1.0	Tern
	976	32	15,632	51,840	1.0	
	1344	32	21,520	21,744	1.0	
NTRU [36]	hps2048509	32	699	747	0.9	Tern
	hps2048677	32	931	979	1.0	
	hps4096821	32	1,230	1,278	1.0	
	hrss701	32	1,138	1,186	1.0	
Classic McEliece [37]	348864	32	261,120	176	1,813	*
	460896	32	524,160	236	2,569	
	6688128	32	1,044,992	288	4,082	
	6960119	32	1,047,319	274	4,328	
	8192128	32	1,357,824	288	5,304	
PQ hybrid HPKE Ciphersuites						
ML-KEM & DH-KEM	512 & X25519	32	832	848	1.0	Tern
	768 & X25519	32	1216	1168	1.0	
	1024 & X25519	32	1600	1648	1.0	

Regarding the KEMs:

- In the classical – i.e. pre-quantum – framework, MLS standard recommends KEMs based on Diffie-Hellman on elliptic curves (DHKEMs), such as DHKEM-X25519, DHKEM-X448, DHKEM-P256 or DHKEM-P521. In a DHKEM, one of the public elements exchanged is modeled as the KEM's public key, whereas the other one is considered as its ciphertext. The Key Derivation Functions (KDFs) used within that KEMs are HKDF with hash functions of various fingerprint sizes. Let us note that these KDFs determine the size of the path secret to be encrypted by MLS (cf. [2]), and therefore have some influence on the value of λ.
- When it comes to post-quantum (PQ) algorithms, that are not specified in the MLS standard, we have considered the most promising KEMs arising from the NIST's PQ competition. This process has led to the standardization of Crystals Kyber [38] [33] – under the name ML-KEM [39] – while three other schemes remain studied in the fourth round of the competition as alternatives to Kyber: BIKE [34], HQC [32] and Classic McEliece [37]. We have also looked at some other KEMs from the PQC

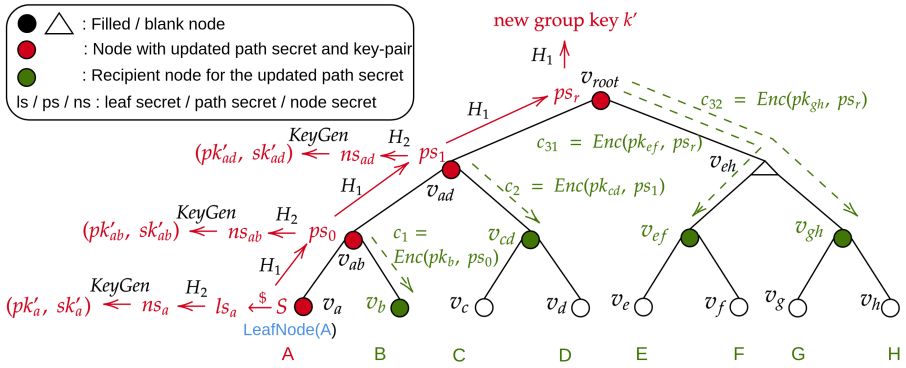


Figure 15: Update, with TreeKEM, of a user's filtered direct path (here user A). This process updates the encryption key-pairs of that user and of all its ancestors; it also generates a new group key.

competition: FrodoKEM [35] and NTRU [36], as well as an hybrid KEM composed of both ML-KEM and DH-KEM-X25519. In this setting, the size of the path secret is of 32 bytes, whatever the PQ KEM considered.

Table 3 underlines the strong difference between the classical framework, where λ varies between 0.4 and 0.6 and the ciphersuites are adapted to binary trees, and the post-quantum one, where most schemes have λ around 0.9-1.0, which fits ternary trees.

Appendix C. Optimization Algorithms

Figure 16 and Figure 17 detail the pseudocodes of our optimized algorithms for a tree evolution mechanism:

- the Lightest Child algorithm, for a user add, with the mechanism to build and update the associated Weighted Ratchet Tree;
- the Bottom Tree Expansion.

Bottom Tree Expansion (T, m)

Input: a tree T of degree m
Output: the expanded tree T'

1 : $\mathcal{L}'_{\mathcal{T}} := \varnothing$
/ Removal of the original leaves from the tree.
2 : $T' := T \backslash \mathcal{L}_{\mathcal{T}}$
3 : **for** $\ell \in \mathcal{L}_{\mathcal{T}}$ **do** :
/ Replacement of the leaves by blank internal nodes.
4 : $T' \mathrel{+}= blankInternNode$
/ Computation of the set of new leaves.
5 : $\mathcal{L}'_{\mathcal{T}} \mathrel{+}= \ell$
6 : **for** $i \in [\![1, m]\!]$ **do** :
7 : $\mathcal{L}'_{\mathcal{T}} \mathrel{+}= blankLeaf$
8 : $T' \mathrel{+}= \mathcal{L}'_{\mathcal{T}}$
9 : **return** T'

Figure 16: Pseudocode description of the Bottom Tree Expansion algorithm.

Appendix D. Additional Experimental Results

This appendix includes some experimental results that did not fit in the body text.

D.1. Efficiency Analysis of a Binary Tree

Figure 18 gives an example of how the communication costs of TreeKEM and our improved protocol diverge during the natural evolution of a member group. In particular, it highlights that tree expansions with TreeKEM (shown in the figure with vertical orange dashed lines) cause an important communication overhead that remains afterwards for quite a long time. Conversely, our optimized protocol has a cost that remains close to the *ideal* one (i.e. the lower bound), whatever the operation carried out during the evolution pattern of the member group.

D.2. Comparison Between Trees of Various Degrees

Figure 19 hereunder depicts the bound on the communication ratio λ under which a non-full ternary tree is in average more efficient than a quaternary one. This experimentally determined bound belongs to the interval $[1.45, 1.70]$, the precise value of that limit ratio depending on the tree evolution mechanism (TreeKEM's or our optimized one).

We additionally provide in Figure 20 the bound of optimality between binary and ternary *full* trees, determined from Equation (12). Despite the fact that in real use-cases, full trees with a degree $m \geq 3$ are not used, this limit value of λ appears very close to the one determined experimentally between binary and *non-full* ternary trees (cf. Section 4.3.2).

Appendix E. Details on the Use of Convex Hull to Determine Optimal Tree Degrees

We detail hereunder in Figure 21 the steps that lead to the computation of the cost function $\kappa_\lambda(\mathcal{S})$ over a set $\mathcal{S}$ of trees, by determining the lower-left convex hull of the points $\nu(\mathcal{S}) = \{(c_p(T), c_s(T)),\ T \in \mathcal{S}\}$.

Lightest Child (W_T)

Input: a weighted ratchet tree W_T
Output: W_T's'Lightest Child lc and W_T modified by the user add

$v := \text{tree_root}(W_T)$
$W_T(v) += 1$
/ Recursive Lightest Child for internal nodes
while $v \notin \mathcal{L}_{\mathcal{T}}$ **do** :
  $lc := \text{node_lightest_child}(v, W_T)$
  $v := lc$
  $W_T(lc) += 1$
return lc, W_T

Node Lightest Child (v, W_T)

Input: a node v and a weighted ratchet tree W_T
Output: the node's Lightest Child lc_v

$(c_j)_{j\in[\![1,m]\!]} := \text{children}(v, W_T)$
$w_1 := W_T(c_1)$
$w_{min} := w_1$
$lc_v := c_1$
for $j \in [\![2, m]\!]$ **do** :
  $w_j := W_T(c_j)$
  if $w_j < w_{min}$ **then** :
    $w_{min} := w_j$
    $lc_v := c_j$
return lc_v

Weighted Ratchet Tree Building (T, m)

Input: a tree T of degree m
Output: the weighted ratchet tree W_T associated with T

$r := \text{tree_root}(T)$
$w_r, W_T := \text{weight_subtree}(r, T, m)$
return W_T

Weight Subtree (v, T, m)

Input: a node v and a tree T of degree m
Output: the weight subtree W_T^v rooted at node v

/ Case of a filled leaf.
if $v \in \mathcal{L}_{\mathcal{T}}$ **and** $T(v) \neq 0$ **then** :
  $w_v := 1$
/ Case of a blank leaf.
elseif $v \in \mathcal{L}_{\mathcal{T}}$ **and** $T(v) = 0$ **then** :
  $w_v := 0$
/ Case of an internal node.
else :
  $(c_j)_{j\in[\![1,m]\!]} := \text{children}(v, T)$
  for $j \in [\![1, m]\!]$ **do** :
    $w_{c_j}, W_T^{c_j} := \text{weight_subtree}(c_j, T)$
  $w_v := \sum_{j=1}^{m} w_{c_j}$
$W_T^v := (w_v, W_T^{c_1}, \cdots, W_T^{c_m})$
return w_v, W_T^v

Figure 17: Pseudocode description of the Lightest Child algorithm, for an optimized User Add.

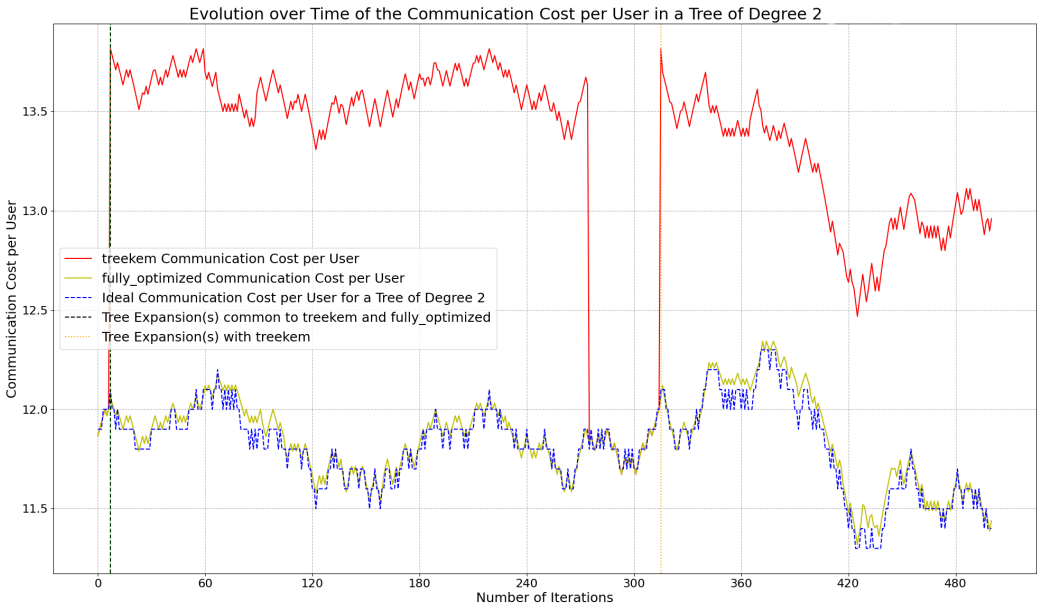


Figure 18: Compared communication costs over time of a Ratchet Tree, between TreeKEM's methodology and our fully-optimized one. The tree expansions undergone by the Ratchet Tree – depending on the evolution mechanism at use – are represented by vertical dashed lines. This figure shows that the communication overhead is mainly induced by tree expansions, especially when this expansion mechanism is not optimized, as with TreeKEM. Improving the user add process then helps "healing" from that overhead; this is not the case either with TreeKEM's left-balanced user add.

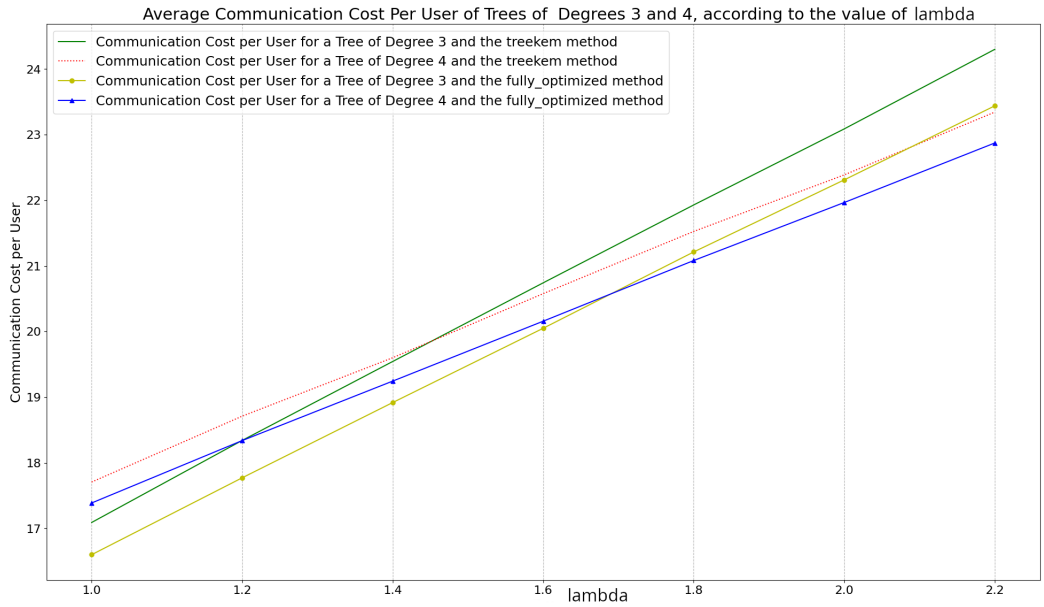


Figure 19: Compared communication costs per user of ternary and quaternary trees evolving with TreeKEM's and our optimized methods. It shows an optimality bound between these two degrees at $\lambda^{3-4} \in [1.45, 1.70]$.

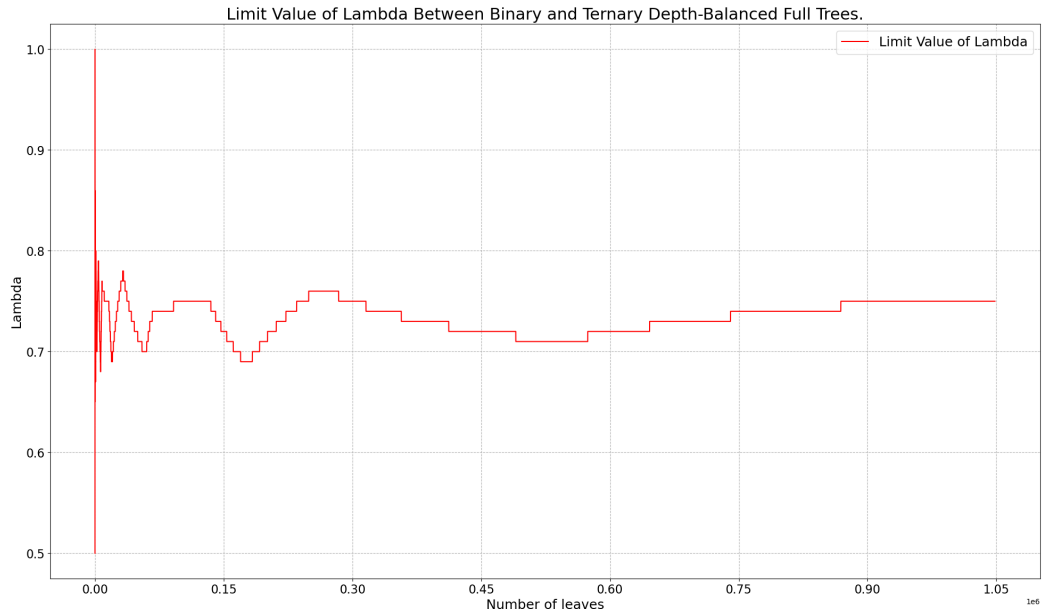
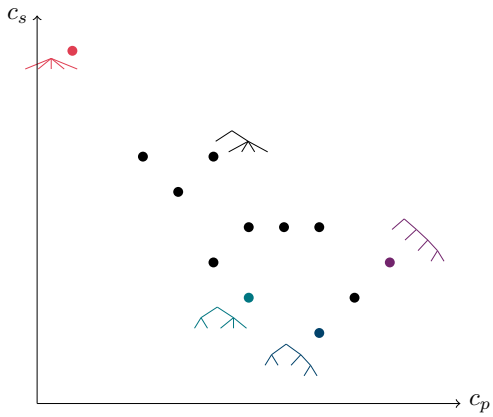
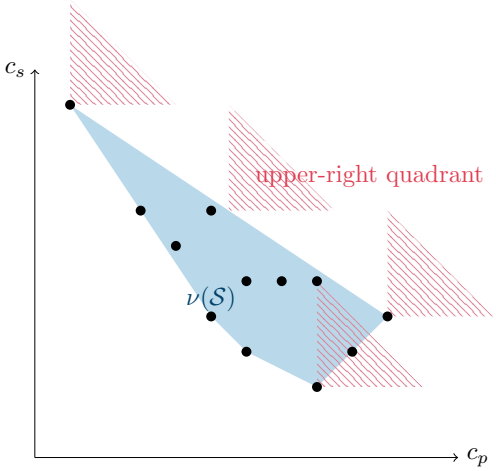


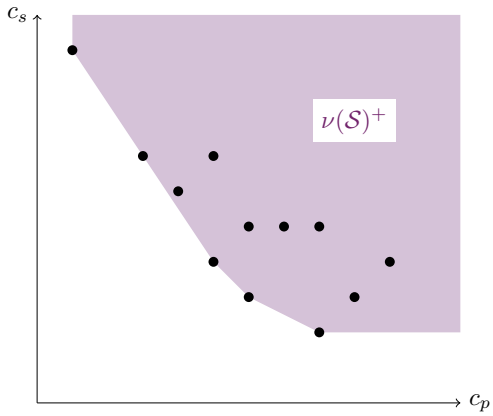
Figure 20: Limit value of the communication ratio λ under which a full binary tree has a better efficiency than a *full* ternary one.



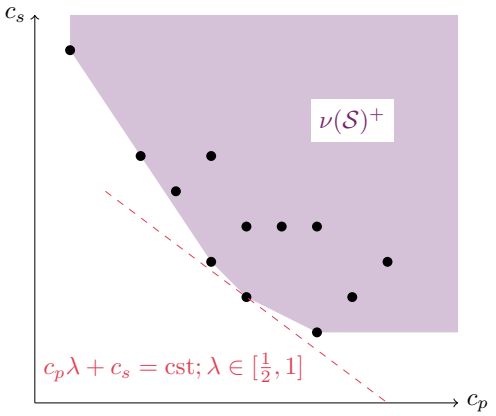
(a) Geometric costs $\nu(\mathcal{S}) = \{(c_p(T), c_s(T)), T \in \mathcal{S}\}$ associated with the set $\mathcal{S}$ of all 12 trees with 5 leaves.



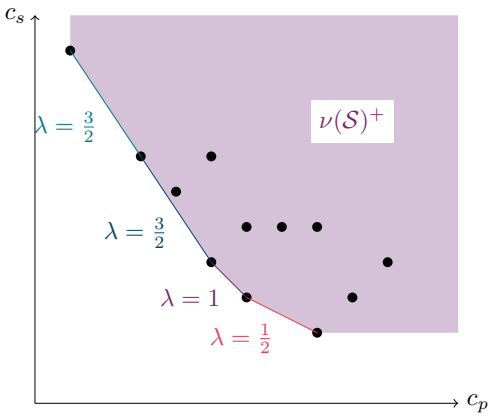
(b) The convex hull $\mathcal{C}$ of $\nu(\mathcal{S})$, and the upper-right quadrant $\mathcal{Q}$, before computing the Minkowski sum.



(c) The Minkowski sum $\mathcal{C}^+ = \mathcal{C} + \mathcal{Q}$; it is also the lower-left convex hull of $\nu(\mathcal{S})$.



(d) The optimal tree for a given communication ratio λ (opposite of the slope) is a point on the lower-left convex hull $\mathcal{C}^+$.



(e) The edges of this polygon also correspond to the limiting values of λ between different optimal trees.

Figure 21: Computation of the cost function $\kappa_\lambda(\mathcal{S})$ over the set $\mathcal{S}$ of trees with five leaves, by determining the lower-left convex hull of the points $\nu(\mathcal{S}) = \{(c_p(T), c_s(T)),\ T \in \mathcal{S}\}$.