GM$^{\text{MT}}$: A Revocable Group Merkle Multi-Tree Signature Scheme

Mahmoud Yehia, Riham AlTawy, T. Aaron Gulliver

Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, CANADA.

Abstract. G-Merkle (GM) (PQCrypto 2018) is the first hash-based group signature scheme where it was stated that multi-tree approaches are not applicable, thus limiting the maximum number of supported signatures to 2^{20}. DGM (ESORICS 2019) is a dynamic and revocable GM-based group signature scheme that utilizes a computationally expensive puncturable encryption for revocation and requires interaction between verifiers and the group manager for signature verification. In this paper, we propose GM$^{\text{MT}}$, a hash-based group signature scheme that provides solutions to the aforementioned challenges of the two schemes. GM$^{\text{MT}}$ builds on GM and adopts a multi-tree construction that constructs new GM trees for new signing leaves assignment while keeping the group public key unchanged, Compared to a single GM instance which enables 2^{20} signature, GM$^{\text{MT}}$ allows growing the multi-tree structure adaptively to support 2^{64} signatures under the same public key. Moreover, GM$^{\text{MT}}$ has a revocation mechanism that attains linkable anonymity of revoked signatures and has a logarithmic verification computational complexity compared to the linear complexity of DGM. The group manager in GM$^{\text{MT}}$ requires storage that is linear in the number of members while the corresponding storage in DGM is linear in the number of signatures supported by the system. Concretely, for a system that supports 2^{64} signatures with 2^{15} members and provides 256-bit security, the required storage of the group manager is 1 MB (resp. $10^{8.7}$ TB) in GM$^{\text{MT}}$(resp. DGM).

Keywords: Digital signatures, Hash-based signature schemes, Group signature schemes, Post-quantum cryptography, Merkle trees.

1 Introduction

A Group Signature Scheme (GSS) is a signature scheme where N members share one public key and any member is allowed to sign anonymously on behalf of the whole group [19]. Such a scheme designates a group manager that is responsible for setup, revealing the signer's identity when needed, and revoking the membership of group members when required. Group signature schemes are usually adopted by applications in which the signer's identity is required to be maintained private while attaining accountability when required. Relevant applications include vehicle safety communication systems in which authorized vehicles share their status information with other nearby vehicles while keeping their identity private in order to prevent tracking [31]. Remote attestation protocols benefit from group signatures where the identities of the attested platforms should be kept private to thwart dedicated platform vulnerability-based attacks[13]. Other applications of group signatures include e-voting and privacy preserving applications on blockchains [13, 4]. Several group signature schemes have been proposed [17, 18, 13, 11, 29, 30]. However, the security of most of these algorithms rely on the hardness of finding discrete logarithms and factoring in finite groups which are solved by Shor's algorithm in polynomial time and thus, they are not post quantum secure [36].

In 2010, Gordon *et al.* proposed the first post quantum (PQ) lattice-based group signature scheme [23]. Later, several theoretical lattice-based constructions were developed [26, 27, 32, 33, 28]. In 2018, the first lattice-based group signature scheme with an experimental implementation was proposed [20]. Although lattice-based signature scheme candidates have been deemed suitable in the current NIST post-quantum cryptography standardization competition (PQC) [34], their group signature constructions are not efficient [40]. Code-based group signature schemes were introduced as a quantum resilient alternative [2, 3, 22], but they have much larger signature sizes on the order of Megabytes [7]. Moreover, the size of the associated public keys and signatures increases with the number of group members.

Hash-based group signature schemes [21, 14] have recently attracted research interest due to recent advances in the design of stateless hash-based signature schemes and the confidence in their PQ security [9, 6, 10, 1]. In 2018, El Bansarkhani and Misoczki introduced Group Merkle (GM), the first post-quantum stateful hash-based group signature scheme [21]. GM is a one-layer Merkle tree construction which limits the maximum achievable tree height and thus restricts the maximum number of signatures that can be issued by the group under one public key. The authors claimed that multi-tree approaches are not applicable for group hash-based schemes without justification and stated that the required storage for each member is a limiting factor. Dynamic Group Merkle (DGM) is a recent hash-based group signature scheme where the group manager can assign signing keys to group members who have used all their keys and add new group members after the group public key has been generated [14]. Additionally, the group manager stores the indexes of the assigned leaves for each user in order to reveal their identity and revoke their membership when required. Challenges to the practical adoption of DGM such as the fact that a verifier needs to interact with the group manager to ensure the validity of the signature were discussed. Moreover, the revocation mechanism utilizes a puncturable encryption algorithm [37] for membership verification with a computational cost that is linear in the number of revoked signatures of the members. The authors of DGM claim that anonymity of revoked signatures is maintained. However, linkability of revoked signatures is possible if the adversary have two subsequent states of the revocation list. Privacy-preserving group membership revocation for PQ schemes is still an open research problem. The works in [38, 39] enable members revocation without compromising their anonymity or requiring a trusted third party. However, the protocols either have linear proving complexity in the number of revocations or rely on history-dependent accumulators through updated certificates. Camenisch at al. proposed member revocation by periodically updating member credentials in which a specific attribute encodes a validity period [16]. Unfortunately, the technique would place extra effort on the group manager who would be essentially running a periodic updates setup phase. All the aforementioned works are also not quantum secure as they rely on non-interactive discrete logarithm based zero knowledge (zk) proofs. The adoption of zk-based revocation schemes in PQ group signature schemes may be attainable if research on generic PQ zk proofs enable their practical implementation.

Our contributions. The contributions of this work are as follows.

- We propose $\mathsf{GM}^{\mathrm{MT}}$, a hash-based group signature scheme that enables 2^{64} signatures per group public key. It utilizes an adaptively growing multi-tree Merkle approach which periodically creates a new GM tree. Consequently, $\mathsf{GM}^{\mathrm{MT}}$ enables the group members to renew their signing leaves without changing the group public key.
- We introduce a revocation algorithm that maintains the anonymity of revoked members while enabling the linkability of their revoked signatures. $\mathsf{GM}^{\mathrm{MT}}$ relies on symmetric encryption and hashing such that the membership verification cost is logarithmic in the number of revoked signatures and the required storage at the group manager is linear in the number of members.
- We provide detailed comparisons between $\mathsf{GM}^{\mathrm{MT}}$ and both GM and DGM. To demonstrate the validity of $\mathsf{GM}^{\mathrm{MT}}$, we implement its procedures using the C language and present the performance in terms of the number of clock cycles.

2 Preliminaries

A Group Signature Scheme (GSS) is a tuple of five polynomial-time algorithms $\mathcal{GS} = (GKGen, GSign, GVerify, GRevoke, GOpen)$, which are given as follows.

- $GKGen(1^n, N)$: The group key generation Alg. takes as inputs the security parameter n and the number of the group members N. It outputs the group public key GPK, the group members secret keys sk_i for $1 \leq i \leq N$, and the group manager secret key sk_{gm} that is used to reveal signer identities.
- $GSign(M, sk_i)$: The group signing Alg. takes as inputs a message M and a group member secret key sk_i. It outputs the signature Σ of the message.
- $GVerify(M, \Sigma, GPK, RevList)$: The group verification Alg. is a deterministic algorithm that takes as inputs a message M and the corresponding signature Σ, the group public key GPK, and the revocation list $RevList$. It outputs 1 for a valid signature and 0 otherwise.
- $GRevoke$: The revocation Alg. updates the revocation list based on the revoked members/signatures to revoke their ability to generate valid signature.
- $GOpen(\Sigma, sk_{gm})$: The open Alg. takes as input the signature Σ and the group manager secret key sk_{gm}, and outputs the identity of the signer.

In what follows, we provide definitions of the standard security notions for analyzing group signature schemes.

Definition 1 *(Correctness). A group signature scheme $\mathcal{GS}$ with a group public key GPK achieves correctness if for an honest signer with a secret key sk_i*

$$\mathsf{Pr}[GVerify(GSign(M, sk_i), M, GPK) = 0] < negl(n)$$

Other notions that capture the required GSS security include unforgeability, anonymity, unlinkability, collusion resistance, exculpability, and framing resistance. It was shown in [8] that full-anonymity and full-traceability ensures that a given GSS achieves all the aforementioned security requirements. The notion of full-anonymity [8] is very strong as it assumes that an adversary has access to the secret keys of all members and the group manager. Camenisch and Groth introduced a relaxed type of anonymity in which an adversary cannot corrupt the group manger and at least two group members, i.e., challenge identities in the anonymity experiment in Figure 1. In our scheme, we follow the anonymity notion introduced by Camenisch and Groth [15] because in our scheme, only secret keys of the group manager are used to reveal signer identities, and knowledge of the signing keys along with the associated signatures also uncovers the corresponding identities. Such a security notion is formally defined in $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}(n, N)$ in Figure 1. Hence in our analysis, we focus on the anonymity and full-traceability security definitions. In their security experiments, we assume an adversary is allowed a training phase where they can call the following oracles.

- $Corrupt(id_i)$: The adversary $\mathcal{A}$ has access to all secret keys of member id_i.
- $chal_b(id_0, id_1, M)$: The oracle returns the signature of message M for a randomly chosen group member id_b for $b \in \{0, 1\}$.
- $Sign(M, id_i)$: The oracle returns the signature of a message M for a randomly chosen group member id_i where $1 \leq i \leq N$.
- $Open(\Sigma, GPK, M)$: The oracle returns the identity id_i of the member who issued the valid signature Σ of message M.

Following [21, 14], we present the security definitions and analysis in the classical setting, i.e., PPT adversaries. For quantum security we consider the Quantum Accessible Random Oracle Model (QROM) [12], where all legitimate users and oracles perform classical computations while adversaries have quantum capabilities. Given that the security of $\mathsf{GM}^{\mathrm{MT}}$ relies on the standard assumptions of hash functions, it is assumed that Grover's search algorithm is used to accelerate exhaustive search in an unstructured space. In such a case, a QPT adversary achieves a maximum of quadratic speed over the considered PPT adversary. The work in [25] gives more details on the generic security of hash function security notions with respect to QPT adversaries in QROM.

Anonymity. In the security experiment Exp^{Anon-b} in Figure 1, the adversary $\mathcal{A}$ is allowed a training phase, train, with unrestricted access to both the signing and opening oracles and they have the ability to corrupt some of the group members.

At the end, $\mathcal{A}$ returns an un-queried random message, M and the identities of two uncorrupted members, id_0 and id_1. Then in the challenge phase, challange, $\mathcal{A}$ calls $chal_b(id_0, id_1, M)$ which return the signature on M signed by one of two uncorrupted users i_0 or i_1. $\mathcal{A}$ wins if they are able to identify the signer's identity with a non-negligible advantage.

Definition 2 *(Anonymity [15]). A group signature scheme $\mathcal{GS}$ achieves anonymity if a probabilistic polynomial time (ppt) adversary $\mathcal{A}$ who is not the group manager but has access to the signing and opening oracles and is able to corrupt all but two group members i_0 and i_1, is not able to reveal the identity of the signer when challenged with a signature of a message that is signed by either i_0 or i_1. $\mathcal{A}$ has a negligible advantage in the experiment $Exp_{\mathcal{GS}}^{Anon-b}$, where $b = \{0, 1\}$ denotes the index of the identity of the signer.*

$$Adv_{\mathcal{GS},\mathcal{A}}^{Anon-b}(n, N) = \mid \mathsf{Pr}[Exp_{\mathcal{GS},\mathcal{A}}^{Anon-0}(n, N) = 1] - \mathsf{Pr}[Exp_{\mathcal{GS},\mathcal{A}}^{Anon-1}(n, N) = 1] \mid \leq negl(n)$$

```
$Exp_{\mathcal{GS},\mathcal{A}}^{Anon-b}(n, N)$
  - $b \in \{0, 1\}$
  - $(GPK, sk_{gm}, sk_*) \leftarrow GKGen(1^n, N)$
  - $(id_0, id_1, M) \leftarrow \mathcal{A}^{Sign(\cdot, id_i), Corrupt(\cdot), Open(\cdot, sk_{gm})}(\mathsf{train}, GPK)$
  - $b \leftarrow \mathcal{A}^{chal_b(id_0, id_1, M)}(\mathsf{challenge}, GPK)$
  - **Return** $b$
```

Fig. 1: Anonymity experiment

Full-traceability. This security notion requires that the group manager is always able to reveal the identity of a signer of a valid signature and trace back every signature to the corresponding signer. Moreover, full-traceability ensures that even if an adversary is capable of corrupting some group members, they are not able to generate a valid signature which is traced by the group manager to an uncorrupted member.

Definition 3 *(Full-traceability [8]). A group signature scheme $\mathcal{GS}$ satisfies full-traceability if a ppt adversary $\mathcal{A}$ that is given unrestricted access to the signing and opening oracles and is able to corrupt some of the group members is not able to generate a valid signature which cannot be opened or traced back by the group manager to an uncorrupted member. $\mathcal{A}$ has a negligible advantage in the experiment $Exp_{\mathcal{GS},\mathcal{A}}^{Full-Trace}$ as defined in Figure 2*

$$Adv_{\mathcal{GS},\mathcal{A}}^{Full-Trace}(n, N) = \mid \mathsf{Pr}[Exp_{\mathcal{GS},\mathcal{A}}^{Full-Trace}(n, N) = 1] \mid \leq negl(n)$$

3 GM$^{\mathrm{MT}}$ Hash-Based Group Signature Scheme

GM$^{\mathrm{MT}}$ is a revocable hash-based group signature scheme that is constructed using a multi-tree approach and utilizes a One Time Signing Scheme (OTS) as the underlying signing scheme. It is designed as a generic construction such that any stateful hash-based Merkle signing scheme with an OTS leaves can be used.

```
$Exp_{\mathcal{GS},\mathcal{A}}^{Full-Trace}(n, N)$
  - $(GPK, sk_{gm}, sk_*) \leftarrow GKGen(1^n, N)$
  - $(\Sigma', M') \leftarrow \mathcal{A}^{Sign(\cdot,id_i),Corrupt(\cdot),Open(\cdot,sk_{gm})}(GPK)$
  - **Return** $GVerify(\Sigma', M', gpk) == 1 \wedge GOpen(\Sigma') = \perp$
    or $id_j$(non corrupted $id_j$)
```

Fig. 2: Full-traceability experiment

However, we recommend instantiating $\mathsf{GM}^{\mathrm{MT}}$ with XMSS-T [25], to mitigate multi-target and path attacks. For more details on the security analysis of hash-based group signature schemes instantiated by XMSS-T, the reader is referred to [41]. $\mathsf{GM}^{\mathrm{MT}}$ provides a flexible setup phase where the group manager generates the group public key independent of the parameters of the group members (OTS public keys and their indexes). Figure 3 shows that $\mathsf{GM}^{\mathrm{MT}}$ can be regarded as a hybrid construction that encompasses several Group Merkle (GM) signature trees (denoted by clusters) at layer 0, and one stateful hash-based signature scheme consuming all higher layers, i.e., layers 1 to $d-1$. Each GM tree at layer 0 contains a subset of the OTSs of all group members while the multi-tree stateful hash-based signature scheme is used by the group manager to sign the roots of the GM trees at layer 0. The group public key, GPK, is the root of the top layer tree which is generated using the group manager's secret key. Such a construction allows layer 0 GM trees to be constructed adaptively as the signing leaves are used up. Specifically, all group members signing leaves are clustered into GM trees where each GM tree has a subset of the signing leaves of all members. This allows the group manager to manage leaf assignment for all members in a clustered manner. Hence, the scheme enables a practical setup phase with less storage requirements for each group member compared to GM [21] because not all the signing leaves for each group member have to be assigned upfront, and a member can reuse the storage that was allocated to their used leaves. In the following, we give detailed specifications of the setup, signing ,verifying, membership revocation, and opening procedures in $\mathsf{GM}^{\mathrm{MT}}$. An algorithmic description of these procedures is provided in Algorithm 1. Table 1 gives the parameters and notation used in the specification of $\mathsf{GM}^{\mathrm{MT}}$.

3.1 Setup Phase and Key Generation

The setup phase is an interactive procedure that involves communication between the group members and group manager for signing leaves assignment. However, since $\mathsf{GM}^{\mathrm{MT}}$ is a multi-tree structure, the group public key is computed by the group manager independent of the inputs from members. Hence, the setup phase is divided into two procedures, group public key generation and signing leaves assignment. The former is performed once during initial group setup while the latter is repeated periodically with the addition of new cluster trees at layer 0.

Key Generation Algorithm. The algorithm randomly samples the secret keys $SK = (sk.enc_{gm}, sk.seed_{gm}) \in \{0,1\}^n \times \{0,1\}^n$, where $sk.enc_{gm}$ is the group

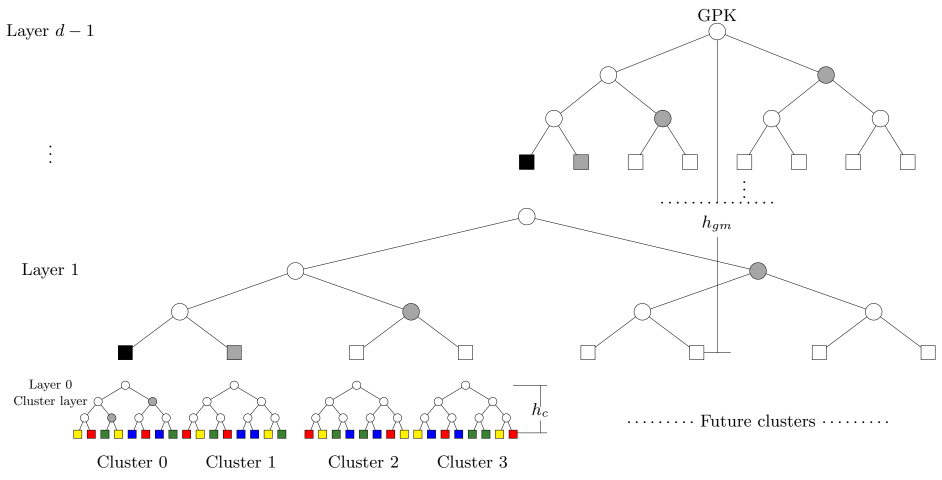

Fig. 3: A simplified Ex. of the $\mathsf{GM}^{\mathrm{MT}}$ initial setup phase. The gray nodes and the first red node in cluster 0 are the auth. path for signing with the first yellow leaf in cluster 0, while the black leaves are the group manager signing leaves.

Table 1: $\mathsf{GM}^{\mathrm{MT}}$parameters and notation.

n	security parameter
N	initial number of group members
B	initial number of signing indexes for each group member
BC_{max}	maximum number of signing leaves for a member in a GM tree (cluster)
Bu_{max}	maximum number of signing leaves for a member in the scheme
d	number of tree layers
h	maximum tree height
h_c	GM/cluster tree height
h_{gm}	group manager tree height, $h_{gm} = h - h_c$
w	Winternitz parameter of the used OTS
l	number of elements, each of length n bits, in the OTS signature
GPK	group public key which is the root of the top layer tree

manager encryption secret key that is used to reveal the signer identity and $sk.seed_{gm}$ is used to generate the trees of the multi-tree signing scheme, e.g., XMSS-T scheme [25], at layers 1 to $d-1$ (the top layer). Each tree has height $h_{gm}/(d-1)$. In an actual instantiation, $sk.seed_{gm}$ may be used in a manner similar to the random secret seed in [25]. The root of the top layer tree is the group public key GPK.

Signing leaves assignment. This procedure adds a new GM cluster tree containing a subset of the signing leaves of all N members to the construction. The trees at layer 0 are GM trees, each of height $h_c = h - h_{gm}$, and the first tree (cluster 0), contains B signing leaves of each group members so there are $NB = 2^{h_c}$ signing leaves in total. Note that each cluster tree contains an equal number of signing leaves for each member. However, $\mathsf{GM}^{\mathrm{MT}}$allows revocation and h_c is a constant, so the number of leaves assigned per member in the i-th cluster, $0 < i < 2^{h_{gm}}$, B, may increase because N may decrease. The assignment procedure is the interactive part of the setup phase and involves the following three steps.

- *Label Assignment.* The group manager sets the maximum number of leaves that can be assigned to a member for the lifetime of the scheme, and assigns to each member a sequence of numbers corresponding to their identity, denoted as labels. Specifically, let $BC_{max} > B$ be the maximum number of leaves that can be assigned to a group member in a cluster, so the maximum number of signatures that a group member can sign is $Bu_{max} = 2^{h_{gm}} \times BC_{max}$. Consequently, the i-th group member is assigned Bu_{max} labels denoted by $b_{0,i}, b_{1,i}, \ldots, b_{Bu_{max}-1,i} = iBu_{max}, iBu_{max}+1, \ldots, iBu_{max}+Bu_{max}-1$ where $0 \leq i \leq N-1$. Since $\mathsf{GM}^{\mathrm{MT}}$ provides member revocation, BC_{max} is chosen to be greater than B to simplify label assignment, so all labels dedicated to a member may not be assigned. Hence, we use the term label to differentiate from a cluster signing leaf index because unlike indexes, not all labels may be assigned. However, each cluster leaf signing index assigned to a member is associated with a label in the dedicated range. Finally, the group manager stores the last assigned label for each group member in the users list, $UList$. Henceforth, the last assigned label of the i-th member is denoted by $la = UList[i]$ and it is used to evaluate their identity by $\lfloor UList[i]/Bu_{max} \rfloor = i$. When a new cluster is being generated, the group manager retrieves the last assigned label, $la = UList[i]$, for each group member, i, and a new range of labels, B, is dedicated to their new cluster signing leaves starting from the next value from the last stored label. More precisely, for a new cluster, the i-th member is given B labels $b_{0,i}, b_{1,i}, \ldots, b_{B-1,i} = UList[i] + 1, UList[i] + 2, \ldots, UList[i] + B$. The group manager then updates the stored label in $UList$ with the last label in the new range, i.e., $UList[i] = UList[i] + B$.

- *Signing keys generation.* Each group member, i, generates B OTS public keys $(pk_{0,i}, pk_{1,i}, \ldots, pk_{B-1,i})$ using their own secret key sk_i, and sends them to the group manager, where $pk_{j,i}$ denotes the j-th public key of the i-th group member within a cluster for $0 \leq i \leq N-1$ and $0 \leq j \leq B-1$.

- *Shuffling and clustering.* The group manager retrieves the last assigned label for each group member and assigns the next set of labels to their public keys (the cluster leaves), in ascending order i.e. $(pk_{0,0}, b_{0,0}), (pk_{1,0}, b_{1,0}), \ldots, (pk_{B-1,0}, b_{B-1,0}), \ldots\ldots, (pk_{0,N-1}, b_{0,N-1}),$
$(pk_{1,N-1}, b_{1,N-1}), \ldots, (pk_{B-1,N-1}, b_{B-1,N-1})$, where $pk_{j,i}$ is the j-th public key of group member i, and $b_{j,i}$ is the corresponding label for $0 \leq i \leq N-1$ and $0 \leq j \leq B-1$. The group manager then updates the last assigned label for each member.

Let $E(k, M)$ denote a symmetric encryption of a plaintext M using the key k. The group manager encrypts the labels assigned to the members by $sk.enc_{gm}$ and generates the corresponding encrypted labels $(Eb_{0,0}, \ldots, Eb_{B-1,0}), \ldots\ldots, (Eb_{0,N-1}, \ldots, Eb_{B-1,N-1})$, where $Eb_{j,i} = E(sk.enc_{gm}, b_{j,i})$. The group manager then generates the cluster leaves,

$L_{0,0}, L_{1,0}, \ldots, L_{B-1,0}, \ldots\ldots, L_{0,N-1}, L_{1,N-1}, \ldots, L_{B-1,N-1}$, by hashing the concatenation of each group member public key and its corresponding encrypted label, i.e. $L_{j,i} = H(pk_{j,i}||Eb_{j,i})$ is the j-th leaf node of group member i. Next, the group manager permutes the group members leaves by reordering their encrypted labels in ascending order. Then, the group manager builds the cluster tree, and signs its root, $root_c$, by the corresponding upper tree leaf node and this continues until the top layer. Finally, the group manager broadcasts to the group members 2^{h_c} tuples of the encrypted labels, cluster tree leaves, and the corresponding signature of its root. Each member, i, identifies their leaf nodes using their public keys, the corresponding encrypted labels, and authentication paths, all of which are referred to by group member parameters, $param_i$.

After a specific time determined by the group manager, by which the group members are expected to have used up almost all their current cluster leaves, the signing leaves assignment procedure is repeated and a new cluster is generated. This is continued until the last cluster is constructed. Figure 3 depicts a simplified example of the initial setup phase. It shows a d layer $\mathsf{GM}^{\mathrm{MT}}$ with 4 clusters in the bottom layer. It is assumed that the group has $N = 4$ members and each member has two signing leaves colored blue, green, yellow, and red in each cluster. $Cluster_1$ is generated after some time period (when the $Cluster_0$ leaves are almost all used up), to provide new signing leaves to the members.

3.2 Signing Algorithm

The signing algorithm takes as input a message M of arbitrary length, the signer's secret key (sk_i), which contains the state of the signer (i-th member), which is the signing index t. The algorithm outputs the signature Σ that contains the OTS signature $\sigma_{OTS,0}$ of the message M and the corresponding authentication path $Auth_0 = (Eb, A_{0,0}, A_{0,1}, \ldots, A_{0,h_c-1})$ from the cluster tree in layer 0. Moreover, Σ contains the signature of the group manager on the cluster root, $\sigma_{root_c} = \sigma_{OTS,1}, Auth_1, \ldots, \sigma_{OTS,d-1}, Auth_{d-1}$, where $\sigma_{OTS,j}$ is the OTS signature of the lower layer tree root, $root_{j-1}$, and $Auth_j$ is the corresponding authentication path $Auth_j = (A_{j,0}, A_{j,1}, \ldots, A_{j,\frac{h_{gm}}{d-1}-1})$. The $\mathsf{GM}^{\mathrm{MT}}$ signature is then given by $\Sigma = \sigma_{OTS,0}, Auth_0, \ldots, \sigma_{OTS,d-1}, Auth_{d-1}$.

3.3 Verification Algorithm

The verification algorithm takes as input the message M, the signature Σ, the public key GPK, and the revocation list $RevList$. It first checks if the received signature has been revoked (see Sec. 3.4). If the signature has not been revoked, the algorithm continues with verification by calculating the OTS public key, pk', from the message digest and the signature element $\sigma_{OTS,0}$ Next, the leaf node is calculated by hashing the concatenation of this OTS public key and the signature element Eb of $Auth_0$, i.e., $L' = H(pk'||Eb)$. Then, the leaf node, L', leaf index, and $(A_{0,0}, A_{0,1}, \ldots, A_{0,h_c-1})$ from $Auth_0$ are used by the Root Computation Algorithm, RCA, (cf. Algorithm 1 in [25] for details) to calculate the cluster root that is used with $\sigma_{OTS,1}$ to get the OTS public key at layer 1. Next, this public key and its index along with the authentication path $Auth_1$ are used to calculate the tree root at layer 1 using RCA. This procedure is repeated until the top layer tree root is calculated, GPK'. If it is equal to the public root, $GPK' = GPK$, the algorithm outputs 1 for a valid signature, and 0 otherwise.

Algorithm 1 $\mathsf{GM}^{\mathrm{MT}}$ Algorithm. Red (resp. blue) denotes the procedures which are performed by the group manager (resp. member) .

```
Setup Phase
Input: $n, N, d, h_{gm}, h_c, BC_{max}$
$(sk.seed_{gm}, sk.enc_{gm}, GPK) \leftarrow GKGen(1^n)$
$Bu_{max} = h_{gm} \times BC_{max}$
for $0 \leq i \leq N-1$ do
    $UList[i] = (i \times Bu_{max}) - 1$
end for
    for $0 \leq i \leq N-1$ do
        $id_i : sk_i \xleftarrow{R} \{0,1\}^n$
    end for
    Return: $sk.seed_{gm}, sk.enc_{gm}, GPK, UList$
```

```
    Cluster Generation
    Input: $N, h_c, UList, sk.enc_{gm}, and\, Bu_{max}$
    for $0 \leq i \leq N-1$ do
        $id_i : (pk_{i,0}, \ldots, pk_{i,B-1}) \leftarrow OTS.KGen(1^n, sk_i)$
    end for
    for $0 \leq i \leq N-1$ do
        $b = UList[i] + 1$
        for $0 \leq j \leq B-1$ do
            $Eb_{i,j} \leftarrow E(sk.enc_{gm}, b+j)$
            $TupleList[iB+j, 0] = (pk_{i,j})$
            $TupleList[iB+j, 1] = (Eb_{i,j})$
        end for
        $UList[i] = b + B - 1$
    end for
    $SortedList \leftarrow sort(TupleList)$
    for $0 \leq p < NB$ do
        $leaf[p] = H(SortedList[p,0] || SortedList[p,1])$
    end for
        $root_c \leftarrow MerkleTree(leaf)$
        $\sigma_{root_c} \leftarrow Sign(root_c, sk.seed_{gm})$
    for $0 \leq i < N$ do
        for $0 \leq j < B$ do
            $p = 0$
            while $p < NB$ do
                if $pk_{i,j} = SortedList[p,0]$ then
                    $param1_i[j,0] = p$
                    $param1_i[j,1] = SortedList[p,1] = Eb_{i,j}$
                    $param1_i[j,2] = Auth$
                    Break
                end if
            end while
        end for
        $param2_i = \sigma_{root_c}$
    end for
```

```
Signing Algorithm
Input: $M, param1_i, param2_i, sk_i, state_i$
$\sigma_{OTS,0} \leftarrow OTS.Sign(M, sk_i, state_i)$
$\mathsf{GM}^{\mathrm{MT}}.\Sigma = M, indx, \sigma_{OTS,0}, Eb, Auth_0, param2_i$
$state_i = state_i + 1$
Return: $\mathsf{GM}^{\mathrm{MT}}.\Sigma$
```

```
Verification Algorithm
Input: $M, GPK, \Sigma, RevList$
if $Eb \in RevList$ then
    Return 0
else
    $pk' \leftarrow OTS.Verify(\sigma_{OTS,0})$
    $L'_{indx} \leftarrow H(pk' || Eb)$
    $root'_c \leftarrow RCA(indx, L', Auth_0)$
    for $1 \leq i \leq d-1$ do
        $L' \leftarrow OTS.Verify(root'_{i-1}, \sigma_{OTS,i})$
        $root'_i \leftarrow RCA(indx, L', Auth_i)$
    end for
    if $root'_{d-1} = GPK$ then
        Return 1
    else
        Return 0
    end if
end if
```

```
Revocation Algorithm
Input: $UList, Bu_{max}, RevList, sk.enc_{gm}, i$
$j = UList[i]$
while $j \geq i \times Bu_{max}$ do
    Add $E(sk.enc_{gm}, j)$ to $RevList$
    $j--$
end while
    $RevList \leftarrow sort(RevList)$
    Return $RevList$
```

```
    Opening Algorithm
    Input: $\Sigma, sk.enc_{gm}, Bu_{max}, N, UList$
    $b' \leftarrow D(sk.enc_{gm}, Eb)$
    if $b' \geq N \cdot Bu_{max} \vee b' > Ulist[\lfloor b'/Bu_{max} \rfloor]$ then
        return $\perp$
    else
        Return $\lfloor b'/Bu_{max} \rfloor$
    end if
```

3.4 Revocation Algorithm

The group manager retrieves the last assigned label of the revoked i-th member, $la = UList[i]$, and then regenerates all the encrypted labels which were assigned to that member, i.e., for the i-th member, the manager generates $E(sk.enc_{gm}, iBu_{max}), E(sk.enc_{gm}, iBu_{max}+1), \ldots, E(sk.enc_{gm}, l_a)$. The generated encrypted labels are added to the revocation list, $RevList$, which is then permuted using a sorting algorithm so that successive entries in the revocation list are not grouped by members.

Revocation Check: The verifier checks if the received signature is revoked or not by first extracting the encrypted label, Eb, from the signature and checking if it exists in the revocation list, $RevList$. If $Eb \in RevList$, then the received signature has been revoked, otherwise the verifier continues the verification steps.

3.5 Opening Algorithm

The opening algorithm takes as input a message M, a signature Σ, and the group manager secret key $sk.enc_{gm}$, and outputs the identity of the signer i. The algorithm first decrypts the signature element Eb to recover the label $b = D(sk.enc_{gm}, Eb)$. Next, the manager calculates the member's identity $i = \lfloor b/Bu_{max} \rfloor$ and checks that b is less than the last assigned label to the i-th group member, $b \leq Ulist[i]$, if not it aborts.

3.6 Recommended Parameters

$\mathsf{GM}^{\mathrm{MT}}$ parameterization follows the NIST PQC requirements which state that a given signing key pair should produce up to 2^{64} signatures while maintaining the claimed security [35]. Thus, we recommend that $\mathsf{GM}^{\mathrm{MT}}$ be instantiated with a four layer ($d = 4$) XMSS-T where the tree height in the bottom layer (clusters), h_c, has three possible values, $h_c = \{16, 18, 20\}$, depending on the number of group members and their signing requirements and storage capabilities. The height of the group manager trees in layers 1 to 3 is 16. The $\mathsf{GM}^{\mathrm{MT}}$ signature size depends on the required security level. More precisely, the $\mathsf{GM}^{\mathrm{MT}}$ signature size is $d \times l + h + 2$ elements, each of length n bits, where n is the security parameter, $n = \{128, 192, 256\}$, and l is the number of OTS signature elements, i.e, XMSS-T utilizes WOTS, then $l = \{35, 51, 67\}$ for the respective aforementioned security parameters [25]. Table 2 gives our recommended parameters for $\mathsf{GM}^{\mathrm{MT}}$ such that it supports at least 2^{64} signatures under the same group public key and the corresponding signature size in bytes (B).

Table 2: $\mathsf{GM}^{\mathrm{MT}}$ recommended parameters and signature sizes.

Instance	bit security	d	h	h_c	h_{gm}	N	B	l	w	Signature (B)
$\mathsf{GM}^{\mathrm{MT}}$-128a	128	4	64	16	48	$2 < N \leq 2^6$	$2^{10} < B \leq 2^{15}$	35	16	3296
$\mathsf{GM}^{\mathrm{MT}}$-128b	128	4	66	18	48	$2^6 < N \leq 2^{10}$	$2^8 < B \leq 2^{12}$	35	16	3328
$\mathsf{GM}^{\mathrm{MT}}$-128c	128	4	68	20	48	$2^{10} < N \leq 2^{16}$	$2^4 < B \leq 2^{10}$	35	16	3360
$\mathsf{GM}^{\mathrm{MT}}$-192a	192	4	64	16	48	$2 < N \leq 2^6$	$2^{10} < B \leq 2^{15}$	51	16	6480
$\mathsf{GM}^{\mathrm{MT}}$-192b	192	4	66	18	48	$2^6 < N \leq 2^{10}$	$2^8 < B \leq 2^{12}$	51	16	6528
$\mathsf{GM}^{\mathrm{MT}}$-192c	192	4	68	20	48	$2^{10} < N \leq 2^{16}$	$2^4 < B \leq 2^{10}$	51	16	6576
$\mathsf{GM}^{\mathrm{MT}}$-256a	256	4	64	16	48	$2 < N \leq 2^6$	$2^{10} < B \leq 2^{15}$	67	16	10688
$\mathsf{GM}^{\mathrm{MT}}$-256b	256	4	66	18	48	$2^6 < N \leq 2^{10}$	$2^8 < B \leq 2^{12}$	67	16	10752
$\mathsf{GM}^{\mathrm{MT}}$-256c	256	4	68	20	48	$2^{10} < N \leq 2^{16}$	$2^4 < B \leq 2^{10}$	67	16	10816

4 Security Analysis

In this section we show that $\mathsf{GM}^{\mathrm{MT}}$satisfies the security requirements of correctness, anonymity [15], and full-traceability [8]. We also analyze the security of the proposed revocation mechanism and discuss the drawbacks of adopting a dynamic approach.

Theorem 1 ***(Correctness)*** *Let GM^{MT} be the multi-tree group Merkle signature algorithm described in Sec. 3. Then GM^{MT} achieves correctness as defined in Definition 1.*

Proof (Sketch). $\mathsf{GM}^{\mathrm{MT}}$utilizes a multi-tree Merkle signing scheme for generating signatures and only uses extra shuffling and clustering procedures to assign the signing leaves to different members. Thus, the correctness of $\mathsf{GM}^{\mathrm{MT}}$ is achieved by the correctness of the underling Merkle signature scheme.

Theorem 2 ***(Anonymity)*** *Let GM^{MT} be the multi-tree group Merkle signature algorithm provided in Sec. 3 with secure hash function H and encryption algorithm E. Then GM^{MT} achieves anonymity for each cluster as defined in Definition 2.*

Proof. We adopt the $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}$ game (see Figure 1) on the group members. The proof follows the strategy in [21]. Assume that each group member is assigned B signing leaves in each cluster, i.e., each group member is assigned a total of $B \times 2^{h_{gm}}$ signing leaves over all clusters. An adversary $\mathcal{A}$ is given access to the signing and opening oracles, and can corrupt some group members. Assume there are only two members i_0 and i_1 that are uncorrupted. Moreover, $\mathcal{A}$ queries the signing and opening oracles for a maximum of $2^{h_{gm}} \times (B-1)$ messages for each uncorrupted member such that the signing oracle replies with $B-1$ signatures from each cluster for the two members, i.e., each member has the ability to sign at least one more message with a leaf from any cluster of the $2^{h_{gm}}$ clusters. Recall that the opening oracle when queried by a signature Σ replies with the decryption of the encrypted label Eb in the signature, $b = D(sk.enc_{gm}, Eb)$, which directly reveals the signing identity i. Thus, $\mathcal{A}$ has $B-1$ labels and their corresponding ciphertext pairs (b_{j,i_g}, Eb_{j,i_g}) for each group member i_g from each cluster where $g = \{0,1\}$ and $Eb_{j,i_g} = E(sk.enc_{gm}, b_{j,i_g})$, $0 \leq j \leq B-2$.

$\mathcal{A}$ queries the signing oracle with an arbitrary message M of their choice such that the signing oracle replies with the signature for either i_0 or i_1. From this signature, $\mathcal{A}$ retrieves the encrypted label Eb_{B-1,i_g}. Moreover, they are able to determine the signing cluster, and thus the corresponding $B-1$ label-encrypted label pairs (b_{j,i_g}, Eb_{j,i_g}), $0 \leq j \leq B-2$, for each group member i_g collected in the query phase. Then, $\mathcal{A}$ is required to correctly guess the identity of the signer. Since the labels for each group member are set sequentially, and $\mathcal{A}$ knows the first $B-1$ labels for each group member, then $\mathcal{A}$ knows with certainty the B-th labels for both group members, i.e., b_{B-1,i_0} and b_{B-1,i_1}. Accordingly, $\mathcal{A}$ must determine which label is the plaintext corresponding to the encrypted label Eb_{B-1,i_g} received in the queried signature. In other words, the adversary needs to win a distinguishability game that distinguishes the encryption of different plaintexts. As the encryption algorithm used is semantically secure, $\mathcal{A}$ has a negligible advantage in winning the $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}$ game.

Theorem 3 ***(Full-traceability)*** *Let GM^{MT} be the multi-tree group Merkle signature algorithm specified in Sec. 3 with secure hash function H, encryption algorithm E, and an underlying existentially unforgeable Merkle signing scheme. Then, GM^{MT} achieves full-traceability as in Definition 3.*

Proof. Recall that the group manager opens a signature by decrypting the encrypted label Eb in the signature. Assume that an adversary $\mathcal{A}$ collects all the signatures from all clusters. i.e., $\mathcal{A}$ knows (Eb_t, pk_t) where pk_t is the OTS public key at leaf index t for $0 \leq t \leq 2^h - 1$. Assuming $\mathcal{A}$ corrupts a set of members $\mathcal{C}$, then $\mathcal{A}$ wins the traceability game $Exp_{\mathcal{GS},\mathcal{A}}^{Full-Trace}$ in Figure 2 if they are successful in either of the following scenarios.

- $\mathcal{A}$ generates a valid signature of the i-th member where $i \in N \wedge i \notin \mathcal{C}$. Since opening a signature depends on the signature element Eb, then $\mathcal{A}$ should include in the signature an element $Eb^\star$ from one of the signatures of any of the uncorrupted members so that it decrypts to a valid label assigned to an uncorrupted member. Furthermore, $\mathcal{A}$ should pair $Eb^\star$ with one of the OTS public keys of a corrupted member so that they can sign using the corresponding secret key. More precisely, $\mathcal{A}$ must find a pair $(pk_{j,i_c}, Eb^\star)$ that is a second preimage of the pair (pk_{j,i_c}, Eb_{j,i_c}), i.e., $H(pk_{j,i_c} || Eb^\star) = H(pk_{j,i_c} || Eb_{j,i_c})$ where pk_{j,i_c} is the j-th OTS public key of a corrupted member i_c and Eb_{j,i_c} is the corresponding encrypted label. The existence of such an adversary contradicts the assumption of a secure hash function. Conversely, $\mathcal{A}$ does not use any of the OTS public keys of the corrupted members, but rather uses some $Eb^\star$ with a forged signature of the underlying Merkle signature scheme such that it passes verification and then decrypts to a valid assigned label. However, this contradicts the existential unforgeability assumption of the underlying signature scheme.
- $\mathcal{A}$ generates a valid signature which the group manager cannot open. In this case, $\mathcal{A}$ includes in the signature an encrypted label Eb' that is not equal to any of the valid encrypted labels which were collected in the query phase. Then following the steps in the previous scenario, $\mathcal{A}$ needs to either pair Eb' with an OTS public key of a corrupted member, or include it with a forgery of the underlying signature scheme. In both cases, the existence of $\mathcal{A}$ contradicts the assumptions of a secure hash function and an existentially unforgeable signing scheme.

4.1 Revocation Security

For revoking a member with identity i, our revocation mechanism updates a revocation list, *Revlist*, by adding the member's encrypted labels that were assigned to their signing leaves, i.e., $Eb_{0,i}, Eb_{1,i}, \ldots, Eb_{la-iBu_{max},i} = E(sk.enc_{gm}, iBu_{max}), E(sk.enc_{gm}, iBu_{max}{+}1), \ldots, E(sk.enc_{gm}, la)$, where $la = Ulist[i]$ denotes the last assigned label. Each of these encrypted labels is part of a signature. Hence, an adversary $\mathcal{A}$ is able to recover the new set of encrypted labels that is added to *Revlist* with updates by comparing the contents of *Revlist* before and after the update. If $\mathcal{A}$ has collected signatures generated by the system before an update of the revocation list, then $\mathcal{A}$ can check if the encrypted labels in some of the collected signatures are in the newly revoked set. Accordingly, if such a set belongs to one revoked member, then $\mathcal{A}$ is able to link these signatures to the same revoked member. Otherwise, the signatures are for more than one revoked member and $\mathcal{A}$ is required to distinguishes the signatures over a small anonymity set (the newly revoked members). In all cases, only the encrypted labels of the revoked members are added to the revocation list, hence, it is infeasible to reveal the identities associated with these labels because they are encrypted. Note that if $\mathcal{A}$ is given only the last updated revocation list, then $\mathcal{A}$ cannot distinguish the newly revoked signatures from the old ones, and hence cannot link a set of signatures to one signer.

Theorem 4 ***(Revocation)*** *Let GM^{MT} be the multi-tree Merkle group signature algorithm provided in Sec. 3 with secure hash function H and encryption algorithm E. Then, GM^{MT} maintains the anonymity of revoked members and linkability of their signatures.*

Proof. Assume an adversary $\mathcal{A}$ has the previous and current states of the revocation list, and a set of signatures that has been collected between two updates of the revocation list. Then, $\mathcal{A}$ is able to recover the set of newly revoked signatures by running the revocation check on the collected signatures against the previous and current states of *Revlist*. If the update of *Revlist* corresponds to revoking one member, then $\mathcal{A}$ is able to link these revoked signatures to this member without revealing their identity. However, if the current states are updated by revoking more than one member, then we adopt an anonymity game for the revoked members which can be seen as a variant of the $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}$ game that allows $\mathcal{A}$ to be challenged with a set of revoked encrypted labels instead of a signature of their choice. $\mathcal{A}$ wins the game if they are able to attribute a subset of the challenge set to a given revoked signer out of two possible revoked signers. Precisely, $\mathcal{A}$ is given access to the opening algorithm for $B-1$ signatures from each cluster signed by each of two newly revoked members, i.e., $\mathcal{A}$ gets $B-1$ (label, encrypted label) pairs from each cluster for each revoked member. Then, they are challenged with the B-th encrypted label from each cluster for each revoked member and are required to determine which encrypted label belongs to which set of $B-1$ (label-encrypted label) pairs. If $\mathcal{A}$ is able to attribute the challenge encrypted labels to a given signer, then they can build another adversary that is able to distinguish between ciphertexts corresponding to a given plaintext, which contradicts the assumption of a secure encryption algorithm.

4.2 Security of dynamic $\mathsf{GM}^{\mathrm{MT}}$

Our scheme can be adapted to allow adding new members at each cluster generation. In this case, the number of leaves assigned to each group member decreases because the maximum number of leaves in a cluster is 2^{20} and the number of group members is increased. A drawback of dynamic $\mathsf{GM}^{\mathrm{MT}}$ is that the anonymity game cannot be played on all clusters. More precisely, if the two challenge identities in $Exp^{Anon-b}_{\mathcal{GS},\mathcal{A}}$ given in Figure 1 are for a newly joined member and an older member, then the game must be played on the clusters which contain signing leaves for both members. This is because if $\mathcal{A}$ is given a signature from clusters created before the new member has joined the group, then $\mathcal{A}$ can determine that this signature is signed by the older member. On the other hand, if the signature comes from clusters created with both members, the anonymity security is the same as that for static group construction given in Theorem 2.

5 Comparison with GM and DGM

In this section, we compare $\mathsf{GM}^{\mathrm{MT}}$ with the hash-based group signature schemes GM [21] and DGM [14]. Due to the multi-tree construction, $\mathsf{GM}^{\mathrm{MT}}$ has a larger signature size than either that of GM or DGM, for example, for 256-bit security, the signature size of $\mathsf{GM}^{\mathrm{MT}}$ instance of largest signature size, $\mathsf{GM}^{\mathrm{MT}}$-256c, is 10.816 KB whereas that of GM (resp. DGM) is 2.88 KB (resp. 2.72 KB).

5.1 $\mathsf{GM}^{\mathrm{MT}}$ and GM

Unlike GM, $\mathsf{GM}^{\mathrm{MT}}$ provides a revocation algorithm and is a multi-tree Merkle construction. Both schemes require comparable computations from the group manager for the opening and setup phase. Hence, we focus on their maximum number of signing leaves and the storage requirements for each group member.

Maximum number of signing leaves. GM is a one layer tree with a static group construction and the maximum number of signing leaves has been stated to be 2^{20} [21]. On the other hand, $\mathsf{GM}^{\mathrm{MT}}$allows the multi-tree structure to grow once the initial signing leaves are consumed by repeating the last two steps of the setup phase. Thus, the group members renew their signing keys each time a new cluster is generated while keeping the group public key unchanged. For a 4-layer $\mathsf{GM}^{\mathrm{MT}}$construction, up to 2^{64} signing leaves are created for the group depending on the tree height h.

Member storage requirements. In GM, the storage required for each group member is reported to be $B(1 + \log N)$ nodes [21]. Note that since the first node of each authentication path and each leaf node contains an OTS public key and an AES-256 ciphertext, the required storage is in fact $B(3 + \log N)$ n-bit elements. In $\mathsf{GM}^{\mathrm{MT}}$, for a cluster of N members, the required storage is $B(2+\log N)+(d-1)l+h_{gm}$ n-bit elements. More precisely, a member stores the B nodes at the $(\log N)$-th level, each of which is n bits, and $B(1 + \log N)$ n-bit elements for the authentication paths. Note that in GM, a group member stores 3 n-bit values per leaf node, while in $\mathsf{GM}^{\mathrm{MT}}$ a group member stores 2 n-bit values for each leaf node. Additionally, in $\mathsf{GM}^{\mathrm{MT}}$ each group member needs to store the signature of the group manager for the cluster tree root, which is composed of $d-1$ OTS signatures, along with the corresponding authentication paths. Table 3 gives the required storage for each group member in GM and $\mathsf{GM}^{\mathrm{MT}}$. We compare $\mathsf{GM}^{\mathrm{MT}}$ and GM when the total number of supported signatures is 2^{20} for the GM tree and $\mathsf{GM}^{\mathrm{MT}}$ cluster, which is the maximum number of signatures for GM, and with $N = 2^{10}$ group members, so the number of signing leaves for each member is $B = 2^{10}$. We choose $\mathsf{GM}^{\mathrm{MT}}$-256 instances for the comparison as it has the highest storage requirements among all instances. The results show that $\mathsf{GM}^{\mathrm{MT}}$-256c saves at least 5.8% of the required storage compared to GM-256. Note that the values in Table 3 are for 256-bit security where $l = 67$. Thus, the above percentages will increase for lower bit security requirements, i.e., for 128 and 192 bit security with $l = 35$ and 51, respectively. Given the recommended parameters in Table 2, the total required storage for each group member in $\mathsf{GM}^{\mathrm{MT}}$ is $B(2 + \log N) + 3l + 48$ n-bit elements.

Table 3: Group member storage for GM and $\mathsf{GM}^{\mathrm{MT}}$with $N = B = 2^{10}$.

Algorithm	$B = N = 2^{10}$	
	Required storage (number of n-bit elements)	
GM	$B(3 + \log N)$	$2^{10} \cdot 13 = 13312$
$\mathsf{GM}^{\mathrm{MT}}$-256c	$B(2 + \log N) + 3l^{\dagger} + 48$	$2^{10} \times 12 + 3l + 48 = 12537$

† The values are for $l = 67$ and 256-bit security.

5.2 $\mathsf{GM}^{\mathrm{MT}}$ and DGM

Both DGM and $\mathsf{GM}^{\mathrm{MT}}$ are revocable GSSs, but DGM is a dynamic GSS that allows new members to be added to the group after the group public key is generated. Unlike $\mathsf{GM}^{\mathrm{MT}}$, DGM requires interaction between verifiers and the group manager to validate the authentication path for each signature verification. Moreover, the group manager in DGM generates the signing keys for the members and thus can sign on their behalf, so it does not satisfy exculpability [5]. A limitation of our scheme is that all group members simultaneously renew their signing keys periodically. Thus, a group member who has used all their signing leaves cannot renew them before a specific time as they need to wait until the new cluster generation occurs. On the other hand, DGM allows new leaves to be assigned on request. In what follows, we compare $\mathsf{GM}^{\mathrm{MT}}$with DGM with respect to the efficiency of the revocation mechanism.

Revocation efficiency. DGM utilizes symmetric puncturable encryption [37] in its revocation mechanism. With each new revoked member, the group manager punctures the encrypted indexes of the signing leaves of all revoked members. Hence, the group manager is required to store all the indexes assigned to all members. In $\mathsf{GM}^{\mathrm{MT}}$, the corresponding storage required is for the last assigned label of each member because all the encrypted labels assigned to a member can be regenerated from this label. For example, consider a $\mathsf{GM}^{\mathrm{MT}}$-256c instance which has 2^{15} members, supports 2^{64} signatures, and provides 256-bit security. The required storage in $\mathsf{GM}^{\mathrm{MT}}$ (resp. DGM) is $2^{15} \times 2^{8}$= 1 MB (resp. $2^{64} \times 2^{8} \approx 10^{8.7}$ TB). Both schemes have equal sized revocation lists and the revocation computational complexity of the group managers are comparable (linear in the size of the revocation list). However, for a revocation check in DGM, the verifier invokes a hash function for 3 times the number of revoked positions in the revocation list [37]. On the other hand, in $\mathsf{GM}^{\mathrm{MT}}$, the verifier must search for an n-bit signature element (the encrypted label, Eb) in a sorted revocation list, $RevList$, which has logarithmic complexity. Hence, our revocation algorithm reduces the computational complexity for verification compared to DGM. Nevertheless, the revocation list is large, so in Appendix A we provide an alternative revocation mechanism where the size of the revocation list is linear in the number of revoked members. The alternative mechanism is equivalent to traditional revocation by key, and may be suitable for some applications that do not require anonymity of the revoked members.

6 Implementation

In this section, we provide an unoptimized implementation of the main procedures of $\mathsf{GM}^{\mathrm{MT}}$ for the purposes of performance evaluation. This C language implementation uses the XMSS$^{\mathrm{MT}}$/WOTS standard implementation given in RFC 8391 [24], [25] employing SHA2-256 as a hash function, and AES256 for encryption. Shuffling the signing of leaf nodes is done by reordering the leaf nodes in ascending order using the sorting algorithm for 256 bit integers.

Table 4 provides the performance in kilocycles and the corresponding milliseconds when the code is executed on an Intel(R) Core(TM) i5-5200U CPU at 2.20 GHz. The values in the table are the average of 100 runs. This table gives the performance for group public key generation, group member OTS public keys generation, (cluster) label encryption, leaf shuffling, cluster root generation, cluster root signing, signature opening, message signing, and signature verification. The reported numbers are for the three instances $\mathsf{GM}^{\mathrm{MT}}$-256a with $(h_c, N, B) = (16, 2^6, 2^{10})$, $\mathsf{GM}^{\mathrm{MT}}$-256b with $(h_c, N, B) = (18, 2^8, 2^{10})$, and $\mathsf{GM}^{\mathrm{MT}}$-256c with$(h_c, N, B) = (20, 2^{10}, 2^{10})$. Other parameters are possible according to the application and member storage capabilities. A process is performed by a user (U) or the group manager (GM).

Table 4: $\mathsf{GM}^{\mathrm{MT}}$ Performance results in kilocycles (kc) and millisecond (ms).

Process	$\mathsf{GM}^{\mathrm{MT}}$-256a	$\mathsf{GM}^{\mathrm{MT}}$-256b	$\mathsf{GM}^{\mathrm{MT}}$-256c
	$(h_c, N, B) = (16, 2^6, 2^{10})$	$(h_c, N, B) = (18, 2^8, 2^{10})$	$(h_c, N, B) = (20, 2^{10}, 2^{10})$
Public key gen. (GM)	1,245,539,484kc - 566,154.3ms		
OTS public keys gen. (U)	6,147,667kc - 2,794.4ms		
Label encryption (GM)	170,471kc - 77.5ms	680,758kc - 309.5ms	2,721,486kc - 1,237.1ms
Shuffling (GM)	48,436kc - 22.1ms	205,428kc - 93.4ms	854,614kc - 388.5ms
Cluster root gen. †(GM)	3,364,756kc - 1,529ms	13,450,764kc - 6,113ms	53,427,148kc - 24,285ms
Cluster root signing (GM)	33,064kc - 15.1ms		
Message signing (U)	2,957kc - 1.4ms		
Signature verification (U)	12,174kc - 5.6ms	15,124kc - 6.9ms	19,326kc - 8.8ms
Signature opening (GM)	46kc - 0.03ms		

† The Merkle tree is constructed after the leaf nodes have been computed.

7 Conclusion

We proposed $\mathsf{GM}^{\mathrm{MT}}$, a revocable hash-based group signature scheme that addresses some of the challenges identified by the designers of the GM and DGM hash-based group signature schemes. Unlike GM, $\mathsf{GM}^{\mathrm{MT}}$ is a multi-tree construction that allows up to 2^{64} signatures under one group public key. It was shown that $\mathsf{GM}^{\mathrm{MT}}$ saves at least 5.8% of the required storage for each group member compared to GM for an $\mathsf{GM}^{\mathrm{MT}}$-256c instance with 2^{10} group members each assigned 2^{10} signing leaves. Unlike DGM, $\mathsf{GM}^{\mathrm{MT}}$ verification procedures do not require interaction with the group manager. Moreover, the required storage for the group manager in $\mathsf{GM}^{\mathrm{MT}}$ is linear in the number of members, while in DGM it is linear in the total number of signatures supported by the scheme. $\mathsf{GM}^{\mathrm{MT}}$ also reduces the computation complexity of checking revocations from linear in DGM to logarithmic in the size of the revocation list. An analysis of $\mathsf{GM}^{\mathrm{MT}}$ with respect to anonymity [15] and full traceability [8] was given which shows that its security relies on the standard security assumptions of hash functions and symmetric encryption, and the existential unforgeability of the underlying signing scheme. Finally, we compared $\mathsf{GM}^{\mathrm{MT}}$ to both GM and DGM, and presented the performance of its procedures using an unoptimized C implementation.

Acknowledgment. The authors would like to thank the anonymous reviewers for their valuable comments that helped improve the quality of the paper.

References

[1] Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Liu, Y., Miller, C., Moody, D., Peralta, R., Perlner, R., et al. Nistir 8309 status report on the second round of the NIST post-quantum cryptography standardization process. *National Institute of Standards and Technology (NIST), US Department of Commerce* (2020).

[2] Alamélou, Q., Blazy, O., Cauchie, S., and Gaborit, P. A practical group signature scheme based on rank metric. In *WAIFI* (2016), Springer, pp. 258–275.

[3] Alamélou, Q., Blazy, O., Cauchie, S., and Gaborit, P. A code-based group signature scheme. *Des Cod.es Crypt 82*, 1-2 (2017), 469–493.

[4] AlTawy, R., and Gong, G. Mesh: A supply chain solution with locally private blockchain transactions. *Proc. Priv. Enhancing Technol. 2019*, 3 (2019), 149–169.

[5] Ateniese, G., and Tsudik, G. Some open issues and new directions in group signatures. In *FC* (1999), Springer, pp. 196–211.

[6] Aumasson, J.-P., and Endignoux, G. Improving stateless hash-based signatures. In *CT-RSA* (2018), Springer, pp. 219–242.

[7] Ayebie, B. E., Assidi, H., and Souidi, E. M. A new dynamic code-based group signature scheme. In *C2SI* (2017), Springer, pp. 346–364.

[8] Bellare, M., Micciancio, D., and Warinschi, B. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT* (2003), Springer, pp. 614–629.

[9] Bernstein, D. J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., and Wilcox-O'Hearn, Z. SPHINCS: Practical stateless hash-based signatures. In *EUROCRYPT* (2015), Springer, pp. 368–397.

[10] Bernstein, D. J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., and Schwabe, P. The sphincs+ signature framework. In *ACM SIGSAC CCS* (2019), pp. 2129–2146.

[11] Boneh, D., Boyen, X., and Shacham, H. Short group signatures. In *CRYPTO* (2004), Springer, pp. 41–55.

[12] Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., and Zhandry, M. Random oracles in a quantum world. In *ASIACRYPT* (2011), Springer, pp. 41–69.

[13] Boneh, D., and Shacham, H. Group signatures with verifier-local revocation. In *ACM CCS* (2004), pp. 168–177.

[14] Buser, M., Liu, J. K., Steinfeld, R., Sakzad, A., and Sun, S.-F. DGM: A dynamic and revocable group merkle signature. In *ESORICS* (2019), Springer, pp. 194–214.

[15] Camenisch, J., and Groth, J. Group signatures: Better efficiency and new theoretical aspects. In *SCN* (2004), Springer, pp. 120–133.

[16] Camenisch, J., Kohlweiss, M., and Soriente, C. Solving revocation with efficient update of anonymous credentials. In *SCN* (2010), J. A. Garay and R. De Prisco, Eds., Springer, pp. 454–471.

[17] Camenisch, J., and Lysyanskaya, A. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO* (2002), Springer, pp. 61–76.

[18] Camenisch, J., and Lysyanskaya, A. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO* (2004), Springer, pp. 56–72.

[19] Chaum, D., and Van Heyst, E. Group signatures. In *EUROCRYPT* (1991), Springer, pp. 257–265.

[20] Del Pino, R., Lyubashevsky, V., and Seiler, G. Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In *ACM SIGSAC CCS* (2018), pp. 574–591.

[21] El Bansarkhani, R., and Misoczki, R. G-merkle: A hash-based group signature scheme from standard assumptions. In *PQCrypto* (2018), Springer, pp. 441–463.

[22] Ezerman, M. F., Lee, H. T., Ling, S., Nguyen, K., and Wang, H. Provably secure group signature schemes from code-based assumptions. *IEEE Transactions on Information Theory 66*, 9 (2020), 5754–5773.

[23] Gordon, S. D., Katz, J., and Vaikuntanathan, V. A group signature scheme from lattice assumptions. In *ASIACRYPT* (2010), Springer, pp. 395–412.

[24] Hülsing, A., Butin, D., Gazdag, S.-L., Rijneveld, J., and Mohaisen, A. Xmss: Extended Merkle signature scheme. In *RFC 8391*. IRTF, 2018.

[25] Hülsing, A., Rijneveld, J., and Song, F. Mitigating multi-target attacks in hash-based signatures. In *Public-Key Cryptography*. Springer, 2016, pp. 387–416.

[26] Laguillaumie, F., Langlois, A., Libert, B., and Stehlé, D. Lattice-based group signatures with logarithmic signature size. In *ASIACRYPT* (2013), Springer, pp. 41–61.

[27] Langlois, A., Ling, S., Nguyen, K., and Wang, H. Lattice-based group signature scheme with verifier-local revocation. In *PKC* (2014), Springer, pp. 345–361.

[28] Libert, B., Ling, S., Mouhartem, F., Nguyen, K., and Wang, H. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In *ASIACRYPT* (2016), Springer, pp. 373–403.

[29] Libert, B., Peters, T., and Yung, M. Group signatures with almost-for-free revocation. In *CRYPTO* (2012), Springer, pp. 571–589.

[30] Libert, B., Peters, T., and Yung, M. Scalable group signatures with revocation. In *EUROCRYPT* (2012), Springer, pp. 609–627.

[31] Lin, X., Sun, X., Ho, P.-H., and Shen, X. Gsis: A secure and privacy-preserving protocol for vehicular communications. *IEEE Transactions on vehicular technology 56*, 6 (2007), 3442–3456.

[32] Ling, S., Nguyen, K., and Wang, H. Group signatures from lattices: Simpler, tighter, shorter, ring-based. In *PKC* (2015), Springer, pp. 427–449.

[33] Nguyen, P. Q., Zhang, J., and Zhang, Z. Simpler efficient group signatures from lattices. In *PKC* (2015), Springer, pp. 401–426.

[34] NIST. Post quantum crypto project. http://csrc.nist.gov/groups/ST/post-quantum-crypto.

[35] NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals.

[36] Shor, P. W. Algorithms for quantum computation: Discrete logarithms and factoring. In *IEEE SFCS* (1994), Ieee, pp. 124–134.

[37] Sun, S.-F., Yuan, X., Liu, J. K., Steinfeld, R., Sakzad, A., Vo, V., and Nepal, S. Practical backward-secure searchable encryption from symmetric puncturable encryption. In *ACM SIGSAC CCS* (2018), pp. 763–780.

[38] Tsang, P. P., Au, M. H., Kapadia, A., and Smith, S. W. Blacklistable anonymous credentials: Blocking misbehaving users without ttps. In *CCS* (2007), CCS '07, ACM, p. 72–81.

[39] Tsang, P. P., Au, M. H., Kapadia, A., and Smith, S. W. PEREA: Towards practical TTP-free revocation in anonymous authentication. In *CCS* (2008), CCS '08, ACM, p. 333–344.

[40] Yang, R., Au, M. H., Zhang, Z., Xu, Q., Yu, Z., and Whyte, W. Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications. In *CRYPTO* (2019), Springer, pp. 147–175.

[41] Yehia, M., AlTawy, R., and Gulliver, T. A. Security analysis of DGM and GM group signature schemes instantiated with XMSS-T. In *(Insecrypt)* (2021), Springer.

A Alternative Solution for a Large Revocation List

In this section, we provide a solution for the large revocation list of $\mathsf{GM}^{\mathrm{MT}}$which is suitable for some applications that do not require anonymity of revoked members. We propose the following modification to the leaf generation procedure.

- The group manager generates a secret key sk_i^* for each group member, for $0 \le i \le N-1$. This key is different from the group member secret key sk_i that is used to generate the WOTS signing keys.
- The encrypted label in $\mathsf{GM}^{\mathrm{MT}}$ is replaced by the output of hashing the concatenation of the corresponding $WOTS.pk$ and the group member key $A^* = H(WOTS.pk||sk_i^*)$.

The remaining procedures are the same as in $\mathsf{GM}^{\mathrm{MT}}$with the following three differences in the revocation, verification and opening procedures.

- To revoke the j-th member, the group manager adds their key sk_j^* to the revocation list, $RevList$.
- In the verification process, the verifier checks if the calculated WOTS from the signature and keys in the revocation list gives the value A^* in the received signature (which means that the signature has been revoked), if not the verifier continues with the verification.
- In the opening process, the group manager checks which group member's secret key sk_i^* gives the value A^* in the signature $A^* = H(WOTS.pk||sk_i^*)$ for $0 \le i \le N-1$.

Applying the above modification has the following consequences.

- The revocation list size is linear in the number of revoked members, while in $\mathsf{GM}^{\mathrm{MT}}$ it is linear in the number of revoked leaves.
- Revocation does not maintain the anonymity of revoked members.
- The verification complexity is linear in the number of revoked members, while $\mathsf{GM}^{\mathrm{MT}}$ verification has logarithmic computational complexity with respect to the number of revoked leaves.
- The opening complexity is linear in the number of members, while $\mathsf{GM}^{\mathrm{MT}}$ has a constant opening complexity, i.e., one decryption operation.