A Scalable Post-quantum Hash-Based Group Signature

Masoumeh Shafieinejad[1], Navid Nasr Esfahani[1], and Reihaneh Safavi-Naini[2]

[1] University of Waterloo, Canada
Masoumeh@uwaterloo.ca, navid.nasresfahani@uwaterloo.ca
[2] University of Calgary, Canada
rei@ucalgary.ca

Abstract. We present a construction for hash-based one-time group signature schemes, and develop a traceable post-quantum multi-time group signature upon it. A group signature scheme allows group members to anonymously sign a message on behalf of the whole group. The signatures are unforgeable and the scheme enables authorized openers to trace the signature back to the original signer when needed. Our construction utilizes three nested layers to build the group signature scheme. The first layer is key management; it deploys a *transversal design* to assign keys to the group members and the openers, providing the construction with traceability. The second layer utilizes *hash pools* to build the group public verification key, to connect group members together, and to provide anonymity. The final layer is a post-quantum hash-based signature scheme, that adds unforgeability to our construction. We extend our scheme to multi-time signatures by using Merkle trees, and show that this process keeps the scalability property of Merkle-based signatures, while it supports the group members signing any number of messages.

Keywords: Post Quantum Signatures, Hash Based Signatures, Group Signatures, Transversal Designs, $\tau-$traceability

1 Introduction

A group signature scheme — first introduced by Chaum and van Heyst [13] — allows group members to sign messages on behalf of a group. The scheme consists of the following entities: *key issuing authority*, *verifier*, *opener(s)*, and *users (members)*. The key issuing authority assigns keys to users to sign messages. The verifier validates whether the signature is a legitimate group signature. The verification process does so without revealing the identity of the signer. Users' anonymity is achieved by generating a public key that is shared by all group members. The scheme should also provide an opening procedure in which openers has the power to renounce the anonymity of a legitimate group signature and determine the respective signer (traceability). Note that no other person can trace a signature back to the signer (anonymity). Moreover, no entity other than group members is able to generate valid signatures (unforgeability). The scheme by Chaum and van Heyst [13] was followed by many schemes who proposed introducing more requirements such as unlinkability, unforgeability, collusion resistance [19], exculpability [19], and framing resistance [20]. Bellare et al. [8] presented formal definitions of security for group signatures which form the standard security model, capturing all the requirements in full-anonymity and full-traceability. Bellare et al. [14] extended the notions of [8] to dynamic groups as well. Dynamic groups have had several proposals to support adding new members and revoking memberships [15,26,27,30].

1.1 Related Work

There has been several proposals for group signatures schemes, however they are mainly not post-quantum schemes [13,19,20,8,15,26,27,30,16,17,18]. There are three main categories of post-quantum signatures: *i*) Lattice-based signatures [31,32], *ii*) Code-based signatures [33], *iii*) Hash based signatures [1,2,3]. The first two categories rely on computational assumptions that do not have an efficient quantum algorithm, while the third one is an extension of Lamport's [4] one-time signature (OTS) scheme and relies on the one-wayness of hash functions. The first post-quantum group signature scheme was a lattice-based construction proposed by Gordon et al. [12], and continued with the schemes that use lattice structures as in [21] and [22], achieving the shortest signature size in $O(\log N)$, in [23] and [24], where N is the group size. The

post-quantum group signatures rarely include code-based schemes [25]. Hash functions are advantageous over other post-quantum proposals as they provide a simple yet strong tool for one-time signature schemes [6,5]. The security of these schemes is undebateable, since it solely relies on the security of hash functions; which motivated NIST to consider hash-based signatures XMSS [1] and LM-HSS [3] as candidates for post-quantum signature standards. Hash-based OTS's are as well easily expandable to multi-time signatures such as SPHINCS [2], XMSS [1], or LM-HSS [3] using Merkle binary tree [5], with no additional requirement. In the structures based on Merkle tree, each leaf is mapped to a one-time signature. The tree root authenticates the signatures generated by all leaves. Despite the fact that hash functions provide strong one-time and multi-time signatures, it is hard to develop a group signature that solely relies on them due to their carrying less structure than other post-quantum primitives. To the best of our knowledge, there has been just one other proposed construction for hash-based group signature, G-Merkle [34]. To overcome the difficulty of extending a single user hash-based signature to a group signature, G-Merkle exploits the structure of Merkle trees. While in a regular Merkle-based signature the leaves only represent the signatures generated for distinct messages, in G-Merkle the leaves represent signatures generated for distinct messages and distinct group members. Hence if the modified tree can support signing B messages for a group of size N, each group member can sign at most $\lfloor B/N \rfloor$ messages, this prevents the scheme to be compatible with group signatures where the number of messages each user needs to sign changes on-the-fly. In addition, in G-Merkle assigning keys (and their indices in the tree) to group members is done by a group manager, introducing a *randomized* structure of Merkle tree. This modification results in two drawbacks. First, the scheme requires an additional computationally secure element, i.e., a block cipher, to shuffle the key indexes among group members. Second, this fixed structure of Merkle tree does not allow extending the tree or using various methods of authentication path computation. Authors leave this issue as an open problem.

1.2 Our Contributions

To introduce a hash based group signature scheme: *i*) We deploy an information theoretically secure structure namely transversal designs to turn a single-user hash-based one-time signature scheme to a multi-user (group) one-time signature. This structure enables group member to sign messages of behalf of the group and remain anonymous. *ii*) Relying on information theoretically secure structures and hash functions, our scheme remains post-quantum secure. *iii*) It is common in signature schemes literature to depend on a powerful group manager who is in charge of issuing keys to the members capable of opening the signatures. However we believe in the context of hash-based signatures, this dependence introduces a single point of failure to the scheme. To prevent this issue, we separate the key issuing entity from opening authorities. We introduce τ-traceability to distributes the tracing power among multiple openers, where opening a signature σ_M requires τ of the O openers to collaborate together. *iv*) We extend our construction to a multi-time group signature by deploying a Merkle tree. Our group signature does not induce any changes in the Merkle tree structure. Hence, it does not encounter any impediment in benefiting from the state-of-the-art advances in authentication path computations. Some of these benefits are shown in standard hash-based signature schemes [1,3]. In summary, we propose a scalable post-quantum group signature scheme that provides a group of non-colluding members with signature unforgeability, anonymity, and traceability.

Paper Organization- Section 2 introduces the requirements of a group signature and the building blocks of our construction. Section 3 presents our construction for hash-based one-time group signature followed by its extension to a multi-time group signature. Section 4 describes the security metrics and evaluates the schemes security correspondingly. In section 5 we evaluate the performance of the construction, measured in time and memory complexity.

2 Preliminaries

We proposes a construction for hash-based group signatures. In this section we introduce group signatures and the blocks we use for our construction; namely *i*) hash-based signatures, *ii*) hash pools, and *iii*) transversal designs.

2.1 Group Signature

We introduce the definitions and security notions associated to group signature schemes following the work of Bellare et al. [8], which describes a comprehensive set of properties. However, we slightly adapt the definitions and the security notions to match a hash-based group signature scheme. The group signature $\mathcal{GS} = (\mathrm{GK}_g, GSig, GV\mathrm{f}, Open)$ is a scheme composed of the following polynomial-time algorithms:

1. $\mathrm{GK}_g(1^s, 1^n, 1^t)$: The group key generation algorithm is a randomized algorithm that takes as input the security parameter s, the number of openers t, and the parameter n (the group size is $N = n^2$) in unary representation. The GK_g algorithm generates and outputs a group public key gpk, the collection of signing key sets $\{gsk_i | i \in [N]\}$ and the set of opening key sets $\{gok_j | j \in [t]\}$, associated to respectively the i^{th} group member and to the j^{th} opener.
2. $\mathrm{GSig}(gsk_i, M)$: The group signing algorithm takes as input a set of signing keys gsk_i, a message $M \in \{0,1\}^l$, and outputs a group signature σ_M on the message.
3. $\mathrm{GVf}(gpk, M, \sigma_M)$: The deterministic group verification algorithm takes as input the group public key gpk, a message m and a group signature σ_M. It outputs $True$ if the signature is valid and $False$ otherwise.
4. $Open(gok, gpk, M, \sigma_M)$: The group opening algorithm is a deterministic algorithm that on inputs the opener's sets of keys, the group's public verification key, the message M and its signature σ_M, outputs the identity of σ_M's signer.

For the scheme to work properly, two conditions must hold: *i*) The verification must be correct, and *ii*) the opening procedure for all honestly generated signatures must be correct. In other words, for any group member $i \in [N]$, the following two expressions have to hold. The first requirement implies that all honestly generated group signatures must be valid, while the second expression allows the openers to recover the identity of a correctly generated signature.

$$GVf(gpk, m, GSig(gsk_i, m)) = True, Open(gok, gpk, m, GSig(gsk_i, m)) = i \tag{1}$$

Security in Group Signatures The main three security definitions for a group signature scheme are unforgeablility, anonymity and traceablity. Some other desired security features of a group signature scheme are: exculpability, coalition and framing resistance, and outsider-/insider- unlinkability. Section 4 provides an explanation for these properties.

2.2 Hash-based Signature

Constructing digital signatures from a one-way function was first introduced by Lamport [4]. The scheme proceeds as follows to sign a 1-bit message b. Two secrets, sk_0, sk_1, are chosen randomly as signing keys, and their corresponding images, computed under the one way, e.g. hash, function f, $pk_0 = f(sk_0)$ and $pk_1 = f(sk_1)$, form the public verification keys. The signature for message b is then sk_b. Any party can verify the signature by evaluating f on sk_b and comparing the result with pk_b in the public verification key. The scheme is secure, fast and simple; however, it requires long signatures and twice secret/public keys as the message length in bits. There are two main categories of hash-based signatures that provide space efficiency for Lamport's proposal; *i*) Winternitz-based schemes, and *ii*) schemes based on 1-cover free families. We introduce a representative of each category in this paper, and later show the compatibility of our design with either of these schemes.

The Winternitz One-Time Signature Scheme The Winternitz scheme that was first introduced by Merkle [44], is a one-time signature that allows a trade-off between the signature cost and size, depending on the scheme parameter w. Winternitz one-time signature (W-OTS) iteratively applies a hash function on a secret input, whereas the number of iterations is defined by the message to be signed. The scheme works as follows [46,47], to sign an l-bit message digest.

1. Key pair generation. First we choose the Winternitz parameter $w \in \mathbb{N}$, $w > 1$, defining the trade off between signature size and signing time. The signature key consists of t random value of length s bits chosen uniformly with the random distribution,

$$sk = (sk_1, \cdots, sk_t) \xleftarrow{\$} \{0,1\}^{(s,t)} \tag{2}$$

where l is computed as:

$$t_1 = \lceil t/w \rceil, t_2 = \lceil (\lfloor \log_2(\lceil t/w \rceil) \rfloor + 1 + w)/w \rceil, t = t_1 + t_2 \tag{3}$$

The verification key is

$$pk = (pk_1, \cdots, pk_t) = (\mathcal{H}^{2^w-1}(sk_1), \cdots, \mathcal{H}^{2^w-1}(sk_t)) \tag{4}$$

2. Signature generation. To sign an l-bit message $M = (M_1 \cdots, M_{t_1})$ given in base w representation, i.e. $M_i \in \{0, \cdots, w-1\}$ for $i = 1, \cdots, t_1$, W-OTS first calculates the checksum: $C = \sum_{i=1}^{t}(2^w - 1 - M_i)$ which in base w is $C = (C_1, \cdots, C_{t_2})$. The length of the base w representation of C is at most t_2 since $C \leq t_1(2^w - 1)$. Then it sets $B = (b_1, \cdots, b_t) = M||C$. The signature of message M is computed as $\sigma = (\sigma_1, \cdots, \sigma_t) = (\mathcal{H}^{b_1}(sk_1), \cdots, \mathcal{H}^{b_t}(sk_t))$.
3. Signature verification. The verifier first computes the base-w string $B = (b_1, \cdots, b_t)$ as the signer does. Then it checks whether $(\mathcal{H}^{2^w-1-b_1}(\sigma_1), \cdots, \mathcal{H}^{2^w-1-b_t}(\sigma_t)) = (pk_1, \cdots, pk_t)$. The signature is accepted iff the comparison holds.

In this work, we use the original version of W-OTS that requires application of a collision-resistant hash function; since the main focus of this paper is to show how the group signature scheme works, not to optimize the underlying schemes. However for implementation, we recommend variants of W-OTS that alleviate the collision resistance requirement [45] and the optimized versions that provide shorter signatures [46].

Signature schemes based on 1-Cover Free Families A *w-Cover-Free Family* (w-CFF) is a collection of subsets of a set $\mathcal{X}$, such that any subset X in the collection is not a subset of the union of any w subsets that does not include X [32]. 1-CFF is a particular case of a w-CFF, where each subset in the collection is not a superset of another one. To use a w-CFF in a signature scheme, first, a set $\mathcal{X}$ of secret keys is generated. Then a w-CFF over X is fixed. Next, a mapping from the set of possible messages to the w-CFF is defined. Finally, the hash values of the secret keys is released. In order to sign a message m, a signer releases the message and the subset of secret keys corresponding to m. Zaverucha and Stinson [32] prove that using w 1-CFF based signature schemes to sign w messages requires less storage than using a w-CFF based signature scheme. Therefore, in this document, we only focus on 1-CFF based signatures. The authors also mention Lamport scheme [4], Bos and Chuam's scheme [41], and Reyzin and Reyzin's scheme [6] as examples of 1-CFF based signature schemes. The following example illustrates this process.

Example 1. Consider a 1-CFF formed by the 2-subsets of a set of size t. Figure 1 depicts the mapping between messages $m_1, \ldots, m_B$ and the 1-CFF over the set $X = \{sk_1, sk_2, \ldots, sk_t\}$.

$\mathcal{X}$	sk_1	sk_2	sk_3	$\cdots$	sk_t
m_1	•	•		$\cdots$	
m_2		•	•	$\cdots$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
m_B	•			$\cdots$	•
PK	$pk_1 = \mathcal{H}(sk_1)$	$pk_2 = \mathcal{H}(sk_2)$	$pk_3 = \mathcal{H}(sk_3)$	$\cdots$	$pk_t = \mathcal{H}(sk_t)$

Fig. 1: Assigning key subsets to sign messages m_1 to m_{2^l} in 1-CFF based signature scheme

The public key is then defined as $PK = (pk_1, pk_2, \ldots, pk_t)$, where$pk_i = \mathcal{H}(sk_i), i \in [k]$, and published publicly. To sign m_2, the signer will publish m_2 together with sk_2 and sk_3. To verify the signature, one needs to verify that m_2 is mapped to $\{2, 3\}$, then calculate their hash values and compare them to the corresponding public keys, i.e., verifying $\mathcal{H}(sk_2) = pk_2$, and $\mathcal{H}(sk_3) = pk_3$,

2.3 Hash Pools

In order to provide users with anonymity, we use *hash pools*. A hash pool is a set of hash values, i.e., $HP = \{\mathcal{H}^\star(x_i) |\ x_i \in \{0,1\}^{s'}, i \in [N]\}$, where $\mathcal{H}^\star$ is a derivation of $\mathcal{H}$ that generates the public key from the private signing key in the underlying OTS; e.g., in W-OTS, $\mathcal{H}^\star(\cdot) = \mathcal{H}^{(2^w-1)}(\cdot)$ which is the (2^w-1)-th iteration of applying $\mathcal{H}(\cdot)$ on an input. Knowing a private key required to calculate any of the public keys in the the hash pool is a proof of authenticity. Each user should prove its authenticity for all/a subset of the hash pools in the scheme, to anonymously gain verification for its produced signature.

2.4 Transversal Design

This work is inspired by Lee and Stinson's TDKPS proposal of a key pre-distribution schemes. This proposal is based on a combinatorial structure known as a transversal design, whose parameters can be chosen to vary the resilience, connectivity, and storage requirements of the key pre-distribution in sensor networks. We start with a definition of τ-*transversal designs* from [7].

Definition 1. *Let $t, \tau \geq 2$ and $n \geq 1$. A transversal design τ-$TD_{(t,n)}$ is a triple $(\mathcal{X}, \mathcal{G}, \mathcal{B})$ such that the following properties are satisfied:*

- *$\mathcal{X}$ is a set of tn elements called points,*
- *$\mathcal{G}$ is a partition of $\mathcal{X}$ into t subsets of size n called design groups,*
- *$\mathcal{B}$ is a set of t-subsets of $\mathcal{X}$ called blocks,*
- *Any group and any block contain exactly one common point, and*
- *Every $\tau-$tuple of points from τ distinct groups is contained in exactly one block.*

We use the following 2-transversal design throughout this paper to provide the use with a more tangible understanding of the scheme.

Example 2. In this example, we list the design groups, and the blocks of the transversal design $2-TD(4,3)$. Such a design has twelve points; we label the points by the elements of the set:

$$\mathcal{X} = \{\alpha_1, \alpha_2, \alpha_3\ \beta_1, \beta_2, \beta_3,\ \gamma_1, \gamma_2, \gamma_3,\ \delta_1, \delta_2, \delta_3\}$$

The partition $\mathcal{G}$, outputs four design groups of size three each, as:

$$\begin{gathered}\text{Design group [1]}= \{\alpha_1, \alpha_2, \alpha_3\}\\ \text{Design group [2]}= \{\beta_1, \beta_2, \beta_3\}\\ \text{Design group [3]}= \{\gamma_1, \gamma_2, \gamma_3\}\\ \text{Design group [4]}= \{\delta_1, \delta_2, \delta_3\}\end{gathered}$$

And eventually, the transversal design delivers the set $\mathcal{B}$ of nine blocks each of size four as follows:

$$\begin{aligned}B_1 &= (\alpha_1, \beta_1, \gamma_1, \delta_1)\\ B_2 &= (\alpha_1, \beta_2, \gamma_2, \delta_2)\\ B_3 &= (\alpha_1, \beta_3, \gamma_3, \delta_3)\\ B_4 &= (\alpha_2, \beta_1, \gamma_2, \delta_3)\\ B_5 &= (\alpha_2, \beta_2, \gamma_3, \delta_1)\\ B_6 &= (\alpha_2, \beta_3, \gamma_1, \delta_2)\\ B_7 &= (\alpha_3, \beta_1, \gamma_3, \delta_2)\\ B_8 &= (\alpha_3, \beta_2, \gamma_1, \delta_3)\\ B_9 &= (\alpha_3, \beta_3, \gamma_2, \delta_1)\end{aligned}$$

It is important to note the followings about transversal designs. First, the *groups* in a transversal design are just subsets of points. Second, the number of produced blocks in a $TD_{(t,n)}$ is n^2 as stated in Lemma 1. Theorems 1 and 2 from [7] prove that it is possible to construct the transversal design for almost any desired group signature. More precisely, if n is a prime power and $t < n$, there exists a $TD_{(t,n)}$. This result provides assurance regarding the applicability of the proposed scheme in various setups.

Lemma 1. *A τ-$TD_{(t,n)}$ has n^τ blocks. [7]*

Transversal designs are closely related to *orthogonal arrays*. An orthogonal array is defined as follows.

Definition 2. *An $OA_\lambda(s, k, v)$ is an $N \times k$ array, such that for any t coordinates, all possible combinations of values for those are repeated exactly λ times.*

Theorem 1. *"Suppose q is a prime power and $2 \leq t \leq q$. Then there exists an $OA(t, q)$."[7]*

Theorem 2. *"Suppose that $n \geq 2$ and $t \geq 3$. Then the existence of any one of the followinflexivilityg designs implies the existence of the other two designs: A $t - 2MOLS(n)$, an $OA(t, n)$ and a $TD_{(t,n)}$".[7]*

3 Our Scheme:

We start with the procedure of turning a hash-based OTS scheme into a one-time group signature scheme; we explain the high level idea accompanied by an example. Next, we discuss approaches to provide the scheme with more flexibility, e.g., in the number of openers. Finally, we demonstrate how we extend our scheme to a multi-time group signature scheme.

3.1 One-Time Signature's High Level Idea

To build a one-time group signature of the form $\mathcal{GS}$ = (GK_g, GSig, GVf, Open) that satisfies traceability, anonymity, and unforgeability, we utilize three nested layers in our construction: *i*) assigning keys using transversal designs, *ii*) generating a public verification key for the group through utilizing *hash pools*, and *iii*) signing messages using a one-time hash-based signature scheme. However, the ordering on layers does not represent their precedence during the process. Conceptually, we start with the hash-based OTS and turn it into a τ-traceable group signature scheme, through using hash pools and transversal designs. As Fig. 2 shows that the core of our scheme is a signature scheme, which provides unforgeability. The hash pools in the second layer provide anonymity. Finally, the transversal design in the outer layer of our scheme provides traceability. We give a high level description of the algorithms in the group signature scheme $\mathcal{GS}$ = (GK_g, GSig, GVf, Open)and show where in the algorithms the mentioned layers are embedded and how they contribute to security.

In this scheme, we consider $N > 1$ to be the number of users in the group, i.e., potential signers, t to be the total number of keys assigned to a user, τ to be the number of openers needed to identify a signer, o to be the total number of openers, and $\mathcal{H}(\cdot)$ to be a cryptographic hash function with collision-resistance property.

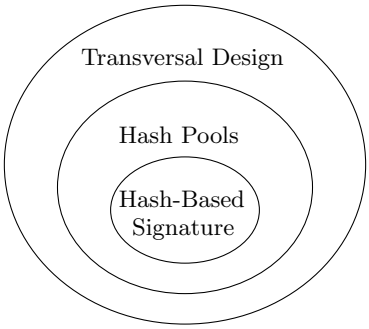

Fig. 2: Three nested layers forming our scheme

One-time Hash-based Signature Scheme- To guarantee unforgeability, our scheme uses an underlying hash-based OTS scheme. The only constraint in the signature scheme is that the signer reveals at least τ secret keys to sign the message M. In this paper, we will show the compatibility of our group signature with two schemes of two main categories of one-time signatures: *i*) 1-CFF OTS, and *ii*) Winternitz signature scheme [5].

Group Verification Key- The hash-based signature scheme generates a set of secret keys gsk_j and a set of corresponding public verification keys gpk_j for each user u_j. We collect the public verification keys of all users — $gpk_j, 1 \leq j \leq N$ — in hash pools to form the group verification key. In what follows, we explain how we use transversal designs to form our hash pools as shown in Figure 3, $HP_i = \{\mathcal{H}(gsk_j^i) \mid j \in [N]\}$. For a message to be verified, the OTS public keys required for its verification should be valid members of their corresponding hash pools.

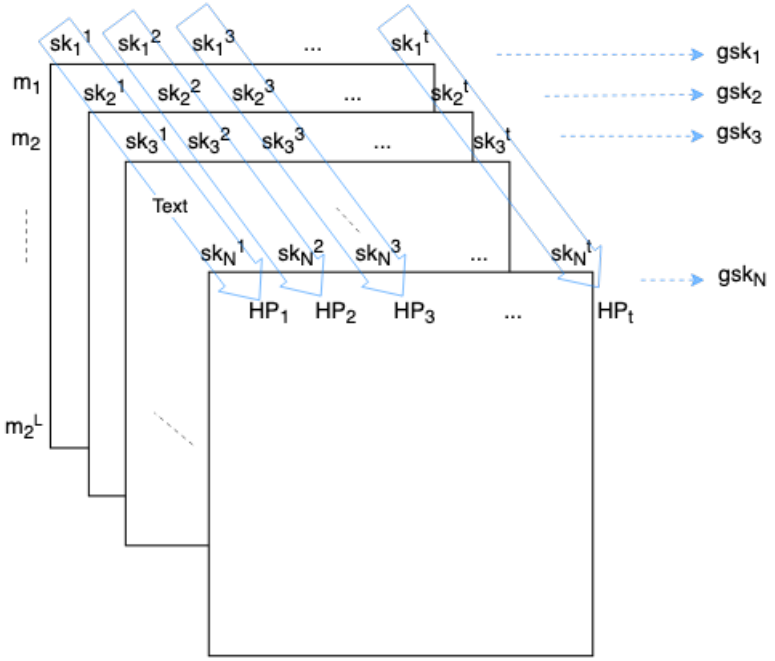

Fig. 3: Using Hash Pools to generate group verification key, HP_i stores the i^{th} elements from each gsk_i in gpk_i for $1 \leq i \leq t$

Transversal Design To make a hash-based signature scheme traceable, we need a structure to enable the openers to link a signature to the key sets of a group member. We utilize transversal designs to provide this structure. As the first step in our construction, we utilize transversal design in the key generation algorithm to assign sets of keys to group members and openers. As described in Section 2.4, a transversal design applies to a set of tn points $\mathcal{X}$, and delivers t design groups and n^τ blocks. We associate the points of a transversal design with the secret keys, the blocks with the group members' keys, and the design groups with the openers' keys in our group signature construction. Hence, the transversal design in Example 2, turns into the following example in a group signature: to distribute keys based on a transversal design, we consider private signing keys to be the points in $\mathcal{X}$. Suppose a hash-based OTS scheme, $\mathcal{S}$, requires a signer to have t private signing keys, and there are $N = n^\tau$ users, then $\mathcal{X}$ needs tn points in total:

$$\mathcal{SK} = \mathcal{X} = \{sk_1^1, sk_1^2, \ldots, sk_1^n, sk_2^1, \ldots, sk_2^n, sk_3^1, sk_3^2, \ldots, sk_t^1, sk_t^2, \ldots, sk_t^n\} = \{sk_i^j \mid i \in [t], j \in [n]\}.$$

As we discussed in Section 2.4, each design group has exactly n elements. Without loss of generality, we assume the partition $\mathcal{G}$ of $\mathcal{X}$ is as follows:

$$\mathcal{G} = \{G_i = \{sk_i^j \mid j \in [n]\} \mid i \in [t]\}.$$

Given this setup, each user u receives the block B_u as the set of private signing keys, gsk_u, and the public key is the collection of t hash pools:

$$gpk = (pk_1, pk_2, \ldots, pk_t), pk_i = HP_i = \{\mathcal{H}^\star(sk_i^j) | j \in [n]\}, i \in [t],$$

Recall that $\mathcal{H}^\star(\cdot)$ is a derivation of $\mathcal{H}(\cdot)$ that generates the public key from the private signing key in the underlying OTS; e.g., in W-OTS, $\mathcal{H}^\star = \mathcal{H}^{(2^w-1)}(\cdot)$ which is the (2^w-1)-th iteration of applying $\mathcal{H}(\cdot)$ on an input. Each opener $O_i, i \in [t]$ receives a set of n pairs, where the first element of a pair is the corresponding public key to sk_i^j from the design group G_i, and the second element is a set of users who have sk_i^j among their keys. The opening key for O_i is gok_i:

$$GOK = \{gok_i \mid i \in [t]\}, gok_i = \{(\mathcal{H}^\star(sk_i^j), \{u \mid sk_i^j \in B_u, u \in [N]\}) \mid j \in [n]\}.$$

The following example demonstrates the key distribution stage for the one-time group signature scheme. We correspond the set of points, $\mathcal{X}$, to the keys we wish to distribute among the openers and the group members in our signature construction. We consider each transversal design group as a set of keys assigned to an opener in our group signature. Subsequently, we associate the blocks to the set of keys assigned to each member of the group in the group signature.

Example 3. In this example, we apply a transversal design 2-$TD(4,3)$ to generate secret keys in the group signature. The design then, outputs the set of keys for the openers and the group members.

$$\mathcal{SK} = \mathcal{X} = \{\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \gamma_1, \gamma_2, \gamma_3, \delta_1, \delta_2, \delta_3\}$$

The partition $\mathcal{G}$, outputs four design groups of size three each, which we map to a collection of key sets, GOK, owned by the openers as follows:

$$\begin{aligned}
&\text{Opener[1]: } \{(\mathcal{H}^\star(\alpha_1), \{1,2,3\}), (\mathcal{H}^\star(\alpha_2), \{4,5,6\}), (\mathcal{H}^\star(\alpha_3), \{7,8,9\})\} = gok_1\\
&\text{Opener[2]: } \{(\mathcal{H}^\star(\beta_1), \{1,4,7\}), (\mathcal{H}^\star(\beta_2), \{2,5,8\}), (\mathcal{H}^\star(\beta_3), \{3,6,9\})\} = gok_2\\
&\text{Opener[3]: } \{(\mathcal{H}^\star(\gamma_1), \{1,6,8\}), (\mathcal{H}^\star(\gamma_2), \{2,4,9\}), (\mathcal{H}^\star(\gamma_3), \{3,5,7\})\} = gok_3\\
&\text{Opener[4]: } \{(\mathcal{H}^\star(\delta_1), \{1,5,9\}), (\mathcal{H}^\star(\delta_2), \{2,6,7\}), (\mathcal{H}^\star(\delta_3), \{3,4,8\})\} = gok_4
\end{aligned}$$

And eventually, the transversal design delivers the set $\mathcal{B}$ of nine blocks each of size four which we map to a vector of key sets, gsk, owned by the group members as follows:

$$\begin{aligned}
gsk_1 &= (\alpha_1, \beta_1, \gamma_1, \delta_1)\\
gsk_2 &= (\alpha_1, \beta_2, \gamma_2, \delta_2)\\
gsk_3 &= (\alpha_1, \beta_3, \gamma_3, \delta_3)\\
gsk_4 &= (\alpha_2, \beta_1, \gamma_2, \delta_3)\\
gsk_5 &= (\alpha_2, \beta_2, \gamma_3, \delta_1)\\
gsk_6 &= (\alpha_2, \beta_3, \gamma_1, \delta_2)\\
gsk_7 &= (\alpha_3, \beta_1, \gamma_3, \delta_2)\\
gsk_8 &= (\alpha_3, \beta_2, \gamma_1, \delta_3)\\
gsk_9 &= (\alpha_3, \beta_3, \gamma_2, \delta_1)
\end{aligned}$$

$$\begin{aligned}
pk_1 = HP_1 &= \{\mathcal{H}^\star(\alpha_j) | j \in [3]\}\\
pk_2 = HP_2 &= \{\mathcal{H}^\star(\beta_j) | j \in [3]\}\\
pk_3 = HP_3 &= \{\mathcal{H}^\star(\gamma_j) | j \in [3]\}\\
pk_4 = HP_4 &= \{\mathcal{H}^\star(\delta_j) | j \in [3]\}
\end{aligned}$$

The transversal design $TD(t,n)$, provides our group signature construction with two interesting properties: *i*) Any opener has the index of each user and the public key corresponding to one of that user's keys exactly once. *ii*) Every pair of group opener keys from distinct openers corresponds to the key set owned by exactly one group member. Subsequently, there is a group of τ openers that can identify the signer correctly.

Signing, Verification, and Opening- To sign a message M, a group member follows the general signing algorithm for the underlying OTS, and publishes the message and its signature $(\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_t})$. In single signer OTS without the anonymity property, the public verification values are calculated for each of the t secret keys. In group signature however, those public verification values of all users are presented in hash pools and are shared among all group members to provide anonymity. Without loss of generality, let $gsk_u^i, i \in [t]$ be the set of secret keys given to user u to sign a message M. To verify a signature $(\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_t})$ on M, the verifier evaluates whether or not the elements of the signature $(\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_t})$ have corresponding hash

values in the hash pools, $\mathcal{H}^\star(gsk_u^{i_1}) \in HP_{i_1}, \mathcal{H}^\star(gsk_u^{i_1}) \in HP_{i_2}, \ldots, \mathcal{H}^\star(gsk_u^{i_1}) \in HP_{i_t}$, or not. To represent the relationship between the signatures and elements of the corresponding hash pool, we introduce new notation. If a signature element σ_j, signed using a private signing key $gsk_{i'}^{j'}$, can be verified by $\mathcal{H}^\star(gsk_{i'}^{j'})$ in the underlying scheme, we write $\sigma_j \bowtie \mathcal{H}^\star(gsk_{i'}^{j'})$. Finally, to open a signature, i.e., to identify the signer, the openers need to use the function $Iden : \mathcal{SK} \times GOK \to \emptyset \cup \binom{[n^\tau]}{n}$. This function takes a private signing key $sk_{i'}^{j'}$ and an opening key set gok_i, and outputs either the $\emptyset$ or a set consisting of n values of u's in $[n^\tau]$ such that gsk_u contains $gsk_{i'}^{j'}$.

$$Iden(\sigma_i, gok_u) = \begin{cases} \emptyset & \text{if } (\mathcal{H}^\star(gsk_{i'}^{j'}), *) \notin gok_u, \sigma_i \bowtie H^\star(gsk_{i'}^{j'}) \\ \{u_1, \ldots, u_n\} & \text{if } (\mathcal{H}^\star(gsk_{i'}^{j'}), \{u_1, \ldots, u_n\}) \in gok_u, \sigma_i \bowtie H^\star(gsk_{i'}^{j'}) \end{cases}$$

Consider α_1 and gok_1 in our Example 3. The function $Iden(\alpha_1, gok_1)$ returns $\{1, 2, 3\}$, which is a set of users who share α_1 with gok_1, namely: $1, 2, 3$. Note the following two properties of the function $Iden(\cdot, \cdot)$.

1. Since the group designs are disjoint, it is only one gok_i that contains sk_i^j. Hence, the output of this function will be $\emptyset$ for all other gok_i's.
2. Let $gsk_u^1, gsk_u^2, \ldots, gsk_u^\tau$ be τ secret keys from user u's group secret key set, gsk_u. For any choice of $i_1, i_2, \ldots, i_\tau \in [t]$, there are exactly τ instances of $gok_{i_h}, h \in [\tau]$ such that each gok_{i_h} intersect with one secret keys gsk_u^i. For such group opener keys, we have $\bigcap_{h=1}^{\tau} Iden(gsk_u^h, gok_{i_h}) = u$. If at least one key is not a member of the corresponding group opener key set, the intersection would result in the empty set. Therefore, this intersection is either the $\emptyset$ or u. We use this property of $Iden(\cdot, \cdot)$ function in opening algorithm in our group signature construction. This shows that there exists at least a set of τ openers who can trace the signature back to the signer.

all the openers (in the next subsection we will discuss the scenarios where it is possible to have only some openers participate) evaluate the function at the released keys and their individual gok_j.

Example 4. Consider the group key distribution from Example 3. Suppose we are using the family of 2-subsets of $[t]$ as the 1-CFF for our signature scheme, as in BC scheme [41]. The group member 1 owns gsk_1, namely, $\{\alpha_1, \beta_1, \gamma_1, \delta_1\}$. If an $s'-$bit message m is associated to the subset $\{1, 2\}$ in the cover-free family, to sign m using these keys, the group member publishes α_1 and β_1, together with m. To verify the signature, any verifier can confirm that $\mathcal{H}^\star(\alpha_1)$ and $\mathcal{H}^\star(\beta_1)$ are members of HP_1 and HP_2, respectively. To identify the signer, any opener $o_i, i \in [t]$ computes $Iden(\alpha_1, gok_i)$ and $Iden(\beta_1, gok_i)$. For the former calculation, the first opener obtains $\{1, 2, 3\}$ and every other opener obtains $\emptyset$; and the latter computation returns $\{1, 4, 7\}$ to the second opener and $\emptyset$ to all other openers. Opener 1 and opener 2 then find the intersection of their results, which reveals the group member 1 is the signer.

3.2 Parameter Flexibility and Generalization

While we can change the value of τ in the transversal design so that the scheme requires cooperation of a different number of openers to identify the signer, this scheme does not provide much flexibility with respect to the total number of openers, and the number of openers needed to identify the signer of a message. Also, despite the fact that, the scheme requires the cooperation of at least two openers to uniquely identify the signer, one opener has enough information to reduce the size of the set of possible signers from n^τ to $n^{\tau-1}$. In the remainder of this section, we will discuss techniques from threshold cryptography that can provide the scheme with more flexibility, in terms of parameters, and also prevents information leak to single opener.

Using Secret Sharing Scheme- Shamir's secret sharing scheme (SSSS) [36] is a well-known method of dividing a secret between members of a group such that no information can be obtained about the secret, unless at least a certain number of members contribute their shares. For a group of n people and a threshold of t, a (t, n)-SSSS can be achieved by constructing a random polynomial $\mathcal{P}(\cdot)$ of degree $t-1$ with constant term equal to the secret. Then, each member receives a distinct pair $(x, \mathcal{P}(x)), x \neq 0$. To reconstruct the secret, t shares are used to find the polynomial $\mathcal{P}(\cdot)$ using Lagrange interpolation. Finally, the secret is

Algorithm 1 Group OTS Scheme $\mathcal{GS} = (GK_g, GSig, GV\text{f}, Open)$ based on Transversal Design

Algorithm $GK_g(1^s, 1^n, 1^t)$
  for $l \leftarrow 1$ to $t \times n$ **do**
    $sk_l \xleftarrow{\$} \{0,1\}^s$
  $\mathcal{SK} = \{sk_1, \ldots, sk_{tn}\}$
  $(\mathbf{gsk}, \mathbf{gok}) = TD_{(t,n)}(\mathcal{SK})$
  for $i \leftarrow 1$ to t **do**
    $HP_i = \{\mathcal{H}^\star(sk_{n(i-1)+1}), \mathcal{H}^\star(sk_{n(i-1)+2}), \ldots, \mathcal{H}^\star(sk_{n(i-1)+n})\}$
  $\mathbf{gpk} = \{HP_1, HP_2, \cdots, HP_t\}$
return $(\mathbf{gsk}, \mathbf{gpk}, \mathbf{gok})$

Key generation algorithm takes the security parameter, number of openers, and square root of the number of group members, and outputs $t \times n$ random secret keys $\mathcal{SK}$. The algorithm then applies $TD_{(t,n)}$ on $\mathcal{SK}$ to obtain the key sets for group members gsk_i's for $1 \leq i \leq n^2$, the key sets for openers gok_j's for $1 \leq j \leq t$, and the group public key gpk is generated.

Algorithm $GSig(gsk_i, M)$
  $(M)_w = (M_1 \cdots, M_{l_1})$
  Checksum: $C = \sum_{i=1}^{l_1}(2^w - 1 - M_i)$, $(C)_w = (C_1, \cdots, C_{l_2})$
  $B = (b_1, \cdots, b_l) = M||C$
  $\sigma = (\sigma_1, \cdots, \sigma_l) = (H^{b_1}(gsk_i^1), \cdots, H^{b_l}(gsk_i^l))$
return σ_M

Member i, calculates the message M and its checksum C in base-w. Then calculates the signature σ.

Algorithm GVf$(\mathbf{gpk}, M, \sigma_M)$
  Flag $= True$
  $(M)_w = (M_1 \cdots, M_{l_1})$
  Checksum: $C = \sum_{i=1}^{l}(2^w - 1 - M_i)$, $(C)_w = (C_1, \cdots, C_{l_2})$
  $B = (b_1, \cdots, b_l) = M||C$
  for $x \leftarrow 1$ to t **do**
    Flag= Flag $\wedge((H^{2^w-1-b_1}(\sigma_1), \cdots, H^{2^w-1-b_l}(\sigma_l)) == (pk_1, \cdots, pk_l))$
  return Flag

The verifier first computes the message M and its checksum C in base-w, to obtain $B = (b_1, \cdots, b_l)$. Then it returns $True$ iff $(H^{2^w-1-b_1}(\sigma_1), \cdots, H^{2^w-1-b_l}(\sigma_l)) = (pk_1, \cdots, pk_l)$.

Algorithm $Open(\mathbf{gok}, \mathbf{gpk}, M, \sigma_M)$
  if GVf$(\mathbf{gpk}, M, \sigma_M)$ = False **then**
    return $\perp$
  else
    for $x \leftarrow 1$ to t **do**
      Opener O_i calculates $X_i = Iden(\sigma_j, gok[i])$
      if $X_i \neq \emptyset$ **then**
        O_i shares X_i with other openers
    Compute the intersection of shared X_i's and call the result u
    return u

Opening algorithm, receives a message and its verified signature. By applying the $Iden(\cdot,\cdot)$ function on any τ secret keys in the signature, the algorithm can identify the signer. The $(Iden(\sigma[j_x], gok[j_x])$ function, returns the identity of all group members who share the secret key $\sigma[j]$ with the opener O_i.

recovered by the evaluation of the polynomial $\mathcal{P}(\cdot)$ at $x = 0$. In this scheme, any subset of $t-1$ or fewer shares does not reveal any information regarding the secret. One important requirement of this scheme is that each share is of the same length of the secret.

To use SSSS in conjunction with our OTGS, one simply applies a $(\tau, o)-$secret sharing scheme to share $\mathcal{G}$ between o openers, such that any τ openers can recover the group designs and identify the signer; while any group of openers with less than τ members cannot learn anything about the signer. When $\mathcal{G}$ is recovered, two group designs are chosen to identify the signer. As it can be seen this method removes the limits on the total number of openers, o instead of t, and on number of openers needed to identify the signer, τ instead of 2. Also, collusion of up to $\tau - 1$ openers does not reduce the size of potential signers because they cannot obtain any information about $\mathcal{G}$. This method requires $o|\mathcal{G}|$, total storage.

Using *t*-AONTs *t-all-or-nothing transforms* (t-AONTs) were introduced by D'Arco et al. [28], as a generalization of Rivest [37] and Stinson [38] work on all-or-nothing transforms. An unconditionally secure t-AONT is a mapping from s input blocks to s output blocks such that the mapping and its inverse can be calculated in polynomial time, and if any t output blocks are missing, no information can be obtained about any subset of t input blocks. Existence of these structures have been studied by Esfahani et al. [29] and Wang et al. [35]. In this scheme, the length of each output block is equal to that of each input block. If we apply a t-AONT on $\mathcal{G}$, it will result in o output blocks, such that any subset of $o - t$ of these blocks does not reveal any information about any pair of the group designs, yielding no information about the signer. This method leaves the limit on the total number of openers, increases the number of openers required to identify the signer, from t to $o - t$. The advantage of this method to secret sharing is that it does not require any extra storage, and the total storage cost is $|\mathcal{G}|$.

3.3 From One-time Signature to Multi-time Signature

We described our one-time signature construction in previous sections. However, as an OTS, its restrain in signing just one message securely, limits its practical use. Therefore, it is also essential to introduce a method to tie our OTS scheme to a multi-time signature (MTS). The common approach for turning a one-time signature scheme to a multi-time one in hash-based signatures literature is using Merkle trees [5] or its variants, as deployed in SPHINCS [2], XMSS [1], or LM-HSS [3]. Hence, we use Merkle trees to extend our scheme as well, keeping the same notation as [1] and [2] in the following elaborations. The Merkle tree, as illustrated in Fig. 4, is a binary tree of height h, with leaves on level 0 that can all be verified by a single value R in level h. We indicate the nodes on level j, $0 \leq j \leq h$, by $Node_{i,j}$, $0 \leq i < 2^{h-j}$. The non-leaf

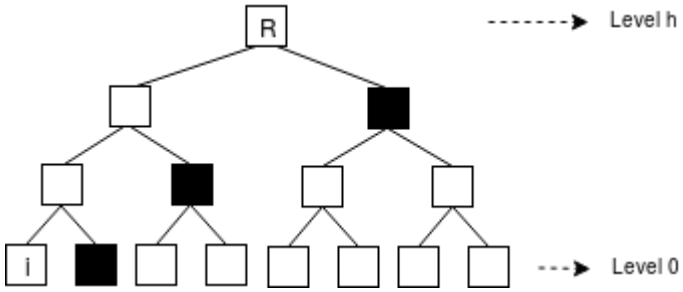

Fig. 4: Merkle tree with authentication path(black nodes) for the i^{th} leaf

nodes, on levels 1 to h, are computed as:

$$Node_{i,j} = \mathcal{H}'(Node_{2i,j-1}||Node_{2i+1,j-1}) \tag{5}$$

where $\mathcal{H}'(\cdot)$ is a collision resistant hash function. However, there are methods to alleviate this requirement; e.g., Buchmann et al. [1] proposes xor-ing public bitmasks with the node values prior to applying the hash

function, which reduces the hash function requirement from collision-resistance to pre-image resistance. To verify a group OTS root in level 0, e.g., $Node_{0,i}$, its authentication path is required. For $Node_{0,i}$,the authentication path is the sequence $Auth = (Auth_0, \ldots, Auth_{h-1})$ of the siblings of all nodes on the path from $Node_{0,i}$ to the root, as shown by black boxes in Figure 4. In what follows we show how we calculate the group OTS root from the OTS public veification keys.

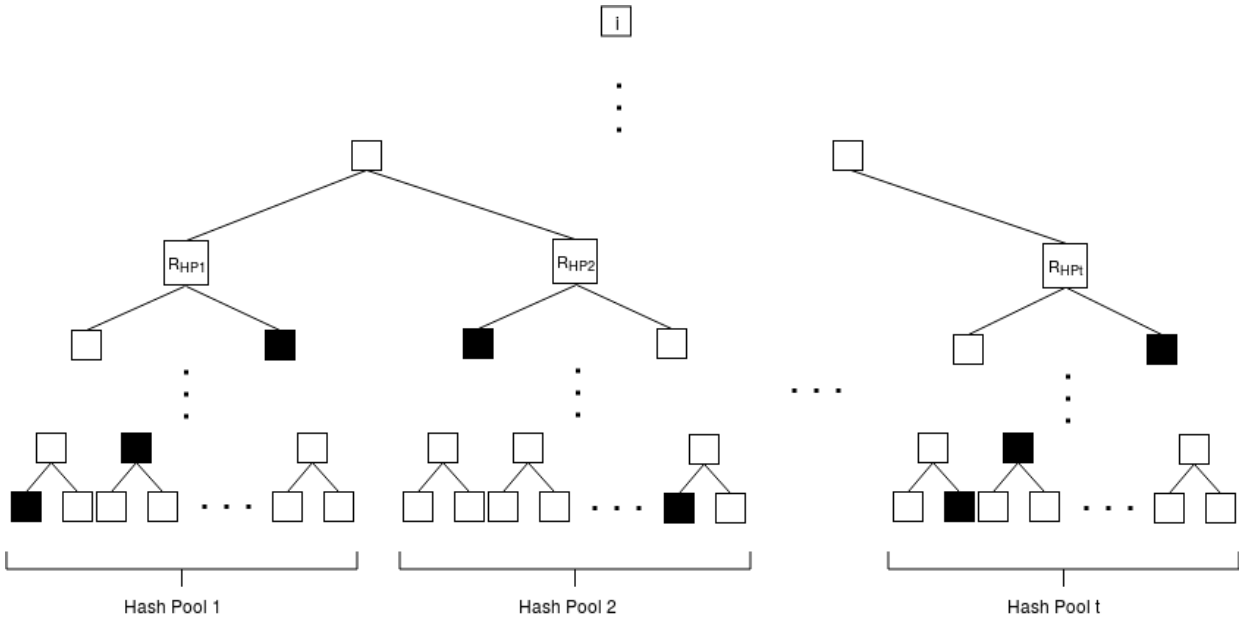

Fig. 5: The Go tree utilized to calculate the value of leaf i in Merkle tree

Grouped One-time Signature (GO) Tree Each leaf in Merkle tree corresponds to a one-time Winternitz signature scheme used to sign a distinct message, e.g., the leaf i in Fig. 5 signs the i^{th} message. To map the group public key in our W-OTS scheme to a single group verification value in node i we use *grouped-OTS (GO) tree*. As described in our OTS algorithm, the secret key in a one-time group signature is a set of tn random values, i.e., $\mathcal{SK} = \{sk_i^j | i \in [t], j \in [n]\}$. The corresponding hash values of these random values are organized in t hash pools of size n to form the group public key, i.e. $gpk = \{HP_i = \{\mathcal{H}^\star(sk_i^j) | j \in [n]\} | i \in [t]\}$. The GO tree is a Merkle tree, storing all users' public verification keys $\mathcal{H}^\star(sk_i^j), j \in [n], i \in [t]$ in level 0 as leaves. The nodes in the higher levels are hash values of the concatenation of their children according to Equation 5. Finally, the public key of the GO tree is placed at GO root and is the node i in the Merkle tree. This approach turns the public key size of the OTS to be just a single hash value. The authentication path for each leaf, similar to in Merkle tree approach, is the path from the leaf to the tree root including the siblings. To be authenticated, the OTS signature needs to include the authentication path per signature element as well as the t elements in σ_M, enabling the root computation given a signature.

Multi-time Signature Scheme Now, we can describe how the signing and verification in the signature with Merkle tree works:

– **Sign.** To sign the i^{th} message, we use the i^{th} GO key pair. The signature $SIG = (i, \sigma, Auth)$, contains the message index i, the GO signature[3] σ, and the Merkle tree authentication path $Auth = (Auth_0, \cdots, Auth_{h-1})$, which is the sibling nodes in the path from the leaf $Node_{i,j}$ to the root R, as shown in Fig. 4.

[3] Recall that the Go signature itself consists of the t signing elements as described in $GSig(gsk_i, M)$ in Algorithm 1 and the authentication path of each of them.

– **Verify.** To verify the signature for the i^{th} message, $SIG = (i, \sigma, Auth)$, the leaf $Node_{0,i}$ of the multi-time signature tree is constructed by building a GO tree on the received W-OTS public keys included in the σ; as described earlier $Node_{0,i}$ is the root of the GO tree. The verifier then computes the sequence $Auth^* = (Auth_0^*, \cdots, Auth_h^*)$. The first node in $Auth^*$ is $Node_{0,i}$ and the rest of the path is calculated using the formula: $Auth_j = \mathcal{H}'(Auth_{j-1}^* || Auth_{j-1})$, for $0 \leq j \leq h$. The verifier accepts the signature, if and only if the final node, $Auth_h$, matches the root R received from the SIG.

Implementation Remarks For an efficient implementation, the dealer gives the GO-tree path to each user, instead of users calculating the path by themselves. It is important that the dealer uses a random permutation of the public keys for each message, so that the pattern of key assigning does not leak information about the signers' identity. Furthermore, as the Merkle tree is the same for each of 2^h messages, each user can calculate the Merkle tree once and store it for its future signing tasks.

4 Security

We first present the definition and description of common security requirements for group signatures, while mentioning the modifications (if needed) to adjust the requirements to our scheme. Next, we prove how our scheme satisfies those requirements.

4.1 Security Evaluation

The main security measurements in group signatures are unforgeability, traceability, and anonymity as Bellare et al. mention in their paper on *Foundations of Group Signatures*[8]. They provide formal definitions for *full-anonymity* and *full-traceability* and prove that these two security requirements cover some other security notions as exculpability, coalition and framing resistance, and outsider-/insider- unlinkability. However, we analyze the performance of the proposed scheme separately in each of the aforementioned security categories, since full-anonymity and full-traceability are not applicable in our hash-based signature. The descriptions of the following security requirements are from Katz and Lindell [9], and Bellare et al. [8].
Notation and Terminology. The scheme is designed for security parameter s, the group size $N = n^\tau$, and the number of openers t. The parameters n and t are used in the transversal design, $TD_{(t,n)}$. Another notation used in the experiments is *st* (for statement), which is the knowledge the adversary $\mathcal{A}$ obtains from one stage (choose) of the experiments and stores to use in another one (guess).

Unforgeability- A basic requirement of any digital signature scheme is that signatures cannot be forged, i.e., it is computationally infeasible to produce message signature pairs (M, σ_M) that are accepted by the verification algorithm, without the knowledge of the secret key(s). In order to define unforgeability, we restrict the adversary so that it does not ask for any secret keys, the adversary can make one query to obtain a digital signature of its choice. In this case a successful attack is producing a valid message signature pair (M, σ_M) such that message M was not queried to the signing oracle [9]. The signature in our construction, consisting of t secret keys, is verified if the keys pass the membership test according to the corresponding hash pools. Thus, for an adversary to forge a signature, it needs to make at least one successful guess on an element of a hash pool. However, due to the security of hash functions, the success chance of a forgery attack is negligible.

Definition. (Unforgeability) *A group signature scheme $GS = (GK_g, GSig, GVf, Open)$ is called unforgeable, if for all probabilistic polynomial adversaries $\mathcal{A}$ and all polynomially bounded n and t the following advantage of the adversary in the experiment $Exp_{GS,\mathcal{A}}^{unforg-cma}(s,n,t)$ is negligible:*
(shown in Algorithm.2)

$$Adv_{GS,\mathcal{A}}^{unforg-cma}(s,n,t) = P[Exp_{GS,\mathcal{A}}^{unforg-cma}(s,n,t) = 1]$$

Algorithm 2 Unforgeability Experiment

Experiment $Exp_{GS,\mathcal{A}}^{unforg-cma}(s,n,t)$
$(\mathbf{gpk},\mathbf{gok},\mathbf{gsk}) \leftarrow GK_g(1^s,1^n,1^t)$
$(st,M_1,\sigma_{M_1}) \leftarrow \mathcal{A}^{Gsig-1(\cdots,M)}(choose,gpk)$
$(M,\sigma_M) \leftarrow \mathcal{A}^{GSig-1(\cdots,M)}(guess,st,M_1,\sigma_{M_1})$
if $\mathrm{GVf}(gpk,(M,\sigma_M))=1 \wedge M \neq M_1$ **then return** 1
else return 0

In unforgeability experiment, the adversary $\mathcal{A}$ runs in two stages: *choose* and *guess*. In *choose*, The adversary $\mathcal{A}$ can query the scheme for the signature of a message (M_1, σ_{M_1}) of its choice, but not more than once, as it is a one time signature. Then in *guess*, $\mathcal{A}$ attempts a forgery by producing a signature σ_M for an arbitrary (but different from M_1) message M. If the signature by $\mathcal{A}$ passes the verification, $\mathcal{A}$ wins the game.

Anonymity- We use the CCA-selfless anonymity definition for our construction as defined in [15]. In this security definition, the adversary has access to the opening oracle, and also to the members' secret keys, with the exception of two group members. These two members are initially chosen by adversary and are used for challenging the adversary later. We use a slightly different version of the CCA-selfless anonymity notion to adjust it with the OTS. We grant the adversary the access to all members' secret keys, except for two identities i_1 and i_2 chosen by the adversary itself. However, regarding querying the opening oracle, we allow the adversary to make queries if not for opening a message signed by i_1 or i_2. This limitation is set due to the fact that a secret key should not be used twice, which is a fundamental property of hash-based signatures. This property makes the signature different from the conventional signatures which rely on public key encryption that is not post-quantum. We denote the adjusted algorithm by $CCA-selfless^*$ and the adversary's access to the opening oracle by $Open^*$. For evaluating anonymity of the scheme, an indistinguishability based method is used. In the method the adversary produces a message and a pair of group-member identities (in "choose" stage), then receives a target signature of the given message under a random one of the two identities. The adversary should have negligible advantage over one-half in determining under which of the two identities the target signature was produced (in "guess" stage). The information passed by the adversary between stages is showed by *st* in the Algorithm 3.

Definition. (CCA Self-less Anonymity) *To provide anonymity, a group signature scheme defined by the algorithms* $GS=(GK_g,GSig,GVf,Open)$ *the following condition should hold: For all probabilistic polynomial adversaries* $\mathcal{A}$ *and all polynomially bounded by* n,t*, the advantage of the adversary in the experiment* $Exp_{GS,\mathcal{A}}^{CPA,anon-b}(s,n,t)$ *is negligible: (shown in Algorithm 3)*

$$Adv_{GS,\mathcal{A}}^{anon-b}(s,n,t) = |P[Exp_{GS,\mathcal{A}}^{CCA-Selfless^*,anon-2}(s,n,t)=1] - P[Exp_{GS,\mathcal{A}}^{CCA-Selfless^*,anon-1}(s,n,t)=1]|$$

Algorithm 3 Anonymity Experiment

Experiment $Exp_{GS,\mathcal{A}}^{CCA-Selfless^*,anon-b}(s,n,t)$
$(gpk,gok,gsk) \leftarrow GK_g(1^s,1^n,1^t)$
$(St,i_1,i_2,M) \leftarrow \mathcal{A}^{Open^*(gok,gpk,\cdot,\cdot)}(choose,gpk,(gsk_{i_2},\cdots,gsk_{i_N}))$
$\sigma_M \leftarrow GSig(gsk_{i_b},M)$
$d \leftarrow \mathcal{A}^{Open^*(gok,gpk,\cdot,\cdot)}(guess,St,\sigma_M)$
if $\mathcal{A}$ did not query its oracle with m, σ_M, i_1, i_2 in the guess stage **then return** d
else return 0

The adversary runs in two stages; *choose* and *guess*. *Choose* stage: the adversary $\mathcal{A}$ is given the access to all gks_i's except gsk_{i_1} and gsk_{i_2}(selfless anonymity[15]), $\mathcal{A}$ can also query opening any signature except the ones generated by i_1 and i_2(CCA adjustment to our scheme). These two identities are chosen by $\mathcal{A}$. *Guess* stage: to challenge the adversary, one of these two identities is chosen randomly, i_b. $\mathcal{A}$ is challenged by m, σ_M generated by i_b. $\mathcal{A}$ should guess b. St is the knowledge $\mathcal{A}$ obtains from *Choose* and stores to use in *Guess*.

Traceability- Traceability enforces that it is not possible to produce signatures which can not be traced to the group member who has produced the signature.

Definition. (Traceability) *A group signature scheme* $GS = (GK_g, GSig, GVf, Open)$ *is called traceable, if for all probabilistic polynomial adversaries* $\mathcal{A}$ *and all polynomially bounded* n *and* t *the following advantage of the adversary in the experiment* $Exp^{trace}_{GS,\mathcal{A}}(s,n,t)$ *is negligible:*
(Shown in Algorithm 4)

$$Adv^{trace}_{GS,\mathcal{A}}(s,n,t) = P[Exp^{trace}_{GS,\mathcal{A}}(s,n,t) = 1]$$

Algorithm 4 Traceability Experiment

Experiment $Exp^{trace}_{GS,A}(s,n,t)$
$(gpk, gok, gsk) \leftarrow GK_g(1^s, 1^n, 1^t)$
$(st, gsk_i) \leftarrow \mathcal{A}^{GSig(gsk_\cdot,\cdot)}(choose, gpk)$
$(m, \sigma_M) \leftarrow \mathcal{A}^{GSig(gsk_\cdot,\cdot)}(guess, st)$
if $\mathrm{GVf}(gpk, (M, \sigma_M)) = 0$ **then return** 0
else if $Open(gok, gpk, M, \sigma_M) = \perp$ **then return** 1
else if $\exists u \in [N]$ s.t. $\{Open(gok, gpk, M, \sigma_M) = j\} \wedge \{u \neq i\}$ **then return** 1
else return 0

In traceability experiment, adversary $\mathcal{A}$ runs in two stages: *choose* and *guess*. In the choose stage, $\mathcal{A}$ chooses an identity i and is given the access to gsk_i. As our scheme is not resistant to members' collusion, we just grant $\mathcal{A}$ that the access to one gsk_i. In the guess stage, $\mathcal{A}$ produces a signature σ_M for a message of its choice, M. If the generated σ_M is traced back to a group member $u \neq i$, or if it is not traced back to a valid group member, $\mathcal{A}$ wins the game.

Other Security Notions

- **Exculpability** Exculpability is the property that no member of the group and not even an opener can produce signatures on behalf of other group members. Exculpability by group members is covered by our definition of traceability. If a member wants to be identifies as another group member, it must have access to at least two secret keys of the other group member that are from different openers. This is a contradiction with our design, as in the underlying transversal design, any two keys from different openers is owned by exactly one group member. Similar proof takes place for an opener who wants to sign on behalf of a group member. The signature property is it has at least two elements from distinct openers. So, an opener being able to sign on behalf of a group member makes a contradiction again, as the opener should have access to one key from another opener as well to do the job, the probability of which is not more than a negligible guess.
- **Coalition Resistance** This is the possibility of a group of signers colluding together to generate signatures that cannot be traced to any of them. The proposed scheme is designed for applications with non-colluding signers though. As a result, two group members can collude together and generate a signature that gets verifies but cannot be traced back to any of them.
- **Framing** Framing is a version of coalition resistance, in which a set of group members combine their keys to produce a valid signatures in such a way that the opening algorithm will attribute the signature to a different member of the group. A strong formalization for framing is the following: consider an experiment in which a group member's identity u is chosen at random from the set of all members, and all group secret keys, except the secret key of u, together with the secret key of the opener are given to the adversary. The adversary wins if it manages to produce a signature which will open as group member u, and a scheme is called secure against framing if no efficient adversary can win with non-negligible probability. Again, as the proposed scheme is designed for applications with non-colluding signers. So, we do not consider this attack in security evaluation.
- **Outsider/Insider Unlinkablility** The intuition is that a party after seeing a list of signatures, can not relate two signatures together as being produced by the same group member. It is necessary to consider distinct security notions for the two cases where the attacker is a group member or it is not, i.e.

insider/outsider unlinkability. This is not an applicable attack to one time signatures. As after signing a message, the whole secrets will be updated; no insider/outsider can relate two signatures together.

4.2 Security of Our Construction

Unforgeability We show that in our construction, unforgeability is guaranteed by the utilized 1-CFF-based signature and hash pools.

Theorem 3. *If $\mathcal{H}(.)$ in the W-OTS signature is a collision resistant hash function, our multi-time signature scheme introduced in Section 3.1 is existentially unforgeable under chosen message attacks.*

Proof. We use the experiment in Algorithm.2 for evaluating unforgeability. According to the experiment, the adversary can query the signature of one message of its choice in OTS. We extend the experiment to multi-time scheme setup by granting adversary access to the scheme's signing oracle 2^h times. Note that the oracle changes the keys after each signature according to our multi-time signature scheme. We first discuss that the unforgeability of our one-time group signature reduces to collision resistance of the hash function in W-OTS signature. Assume the adversary provides a signature for a message of its choice, M_1, after querying the scheme for the signature of a message M. In W-OTS each message is signed by a set of t keys, each element in the signature is verified by the corresponding hash pool. Hence, in order to existentially forge a signature on a new message the forger has to guess the corresponding signature element to at least one elements stored in a hash pool correctly. Therefore, forging a signature in the group signature reduces to forging a signature in the underlying W-OTS scheme which is not possible due to collision resistance of the hash function in the scheme. Now we prove how the unforgeability of the OTS results in the unforgeability of the multi-time signature scheme as well. Assume the attacker $\mathcal{A}$ queries the multi-time signing oracle 2^h times and comes up with a valid signature for a new message of the type $(i, M_i, \sigma_{M_i}, Auth)$. Then the attacker breaks the unforgeability of the OTS. The reason for the claim is even it is assumed that the attacker recovers all tree nodes' values by 2^h query responses, none of that information helps to recover the σ in the OTS. Still the problem of forging the signature σ holds, unless the hash function in the Merkle tree, $\mathcal{H}'$, is not collision resistant. Since forging a signature in our multi-time signature scheme require $\mathcal{H}'$ to not satisfy collision resistance, or the underlying OTS to not be unforgeable, it is not a valid assumption. Therefore, the multi-time signature is unforgeable.

Anonymity We show that in our construction, anonymity is provided by the utilized hash pools and its adjustment with the transversal design.

Theorem 4. *Let $\mathcal{GS}$ be the group signature construction as introduced in Section 3.1, with hash pools forming the group public key and the $TD_{(t,n)}$ be the transversal design utilized in the key generation algorithm in $\mathcal{GS}$. $\mathcal{GS}$ is $CCA-$selfless anonymous.*

Proof. We use the experiment in Algorithm 3 for evaluating anonymity. The adversary $\mathcal{A}$ provides the challenger with a message M and the identities of two group members, u_1 and u_2. $\mathcal{A}$ has access to the secret keys of other users, also it can query the opening oracle but for the signature produced by u_1 or u_2. The challenger chooses one of the identities, u_1 or u_2, randomly, u_b, and provides the signature of the message M by u_b; i.e. $\sigma_M = (gsk^{i_1}_{u_b}, gsk^{i_2}_{u_b}, \cdots, gsk^{i_t}_{u_b})$. The adversary is challenged to guess the real identity of u_b. There are t secret keys in the signature which are not dependant to the b. Hence, no information about b could leak from this step. Secondly, the adversary has access to the verification algorithm, which takes an element $gsk^{i_i}_{u_b}$ as the input, and checks whether it generates a valid element of the corresponding hash pool in $gpk[j]$. This function does not provide any information of other elements stored in the hash pool. The last reason for anonymity is the correspondence of the key sets and the identities of the group members is not known to any entity in the construction. We would like to mention here that even for an opener, the best guess for the identity of the signer is $n^{-(\tau-1)}$ instead of $n^{-(\tau)}$. As described in 3.1, the $Iden(gsk^j_u, gok_{i_j})$ function outputs a set of $n^{-(\tau-1)}$ identities, and if not collaborated with another opener, it cannot identify the signer. Hence, the opener cannot do any better than identifying a set of probable signers by observing the signature by i_b. Therefore, we call our construction $n^{\tau}-$anonymous to a general adversary, and $n^{(\tau-1)}-$anonymous to an opener.

Traceability We show that in our construction, traceability is provided by the utilized transversal design.

Theorem 5. *Let $\mathcal{GS}$ be the group signature construction as introduced in Section 3.1, and $TD_{(t,n)}$ be the transversal design deployed in the construction; $\mathcal{GS}$ is traceable.*

Proof. Suppose there exists a PPT adversary for $\mathcal{GS}$, who proposes a signature that is not traceable to the signer. This translates to there are two secret keys in the signature that trace back to the wrong group member or do not trace back to any group member at all. We prove the impossibility of both cases. Let's consider the signature $\sigma_M = (gsk_u^{i_1}, gsk_u^{i_2}, \cdots, gsk_u^{i_t})$ by group member u for message M, with $gsk_u^1, gsk_u^2, \cdots, gsk_u^\tau$ being traced back to group member u' rather than u or not being traced back to any group member. The first case contradicts the transversal design's properties, as any τ points from different group designs(openers) exist in exactly on block(gsk). This is defined as the second property of the function $Iden(\cdot,\cdot)$ in Section 3.1. Therefore, if there exists an algorithm, that breaks an essential property in the underlying transversal design, it breaks the the second property of the function $Iden(\cdot,\cdot)$. Now, let's consider the case that adversary's offered signature does not trace back to any group member. Again consider the signature $\sigma_M = (gsk_u^{i_1}, gsk_u^{i_2}, \cdots, gsk_u^{i_t})$ with $gsk_u^1, gsk_u^2, \cdots, gsk_u^\tau$ tracing back to no group member. According to GK_g, each secret key assigned to the group member u is shared with a distinct opener. So, $gsk_u^1, gsk_u^2, \cdots, gsk_u^\tau$ are shared with the openers gok_{i_1}, gok_{i_2}, $\cdots$, gok_{i_τ} respectively. Again, according to $Iden(\cdot,\cdot)$ function of the transversal design, $\bigcap_{h=1}^{\tau} Iden(gsk_u^h, gok_{i_h})$ is an index of a block, i.e. the identity of a group member. Hence an algorithm breaking the traceability of our construction by tracing back to no group member's identity, breaks the second property of the function $Iden(\cdot,\cdot)$ of the deployed transversal design.

5 Performance Evaluation

We evaluate our scheme in space and time complexity, and compare it with G-Merkle [34]. Since G-Merkle uses W-OTS, we use W-OTS as well as our underlying OTS to provide a fair comparison. Note that, G-Merkle does not provide the features provided by our scheme, which enables it benefit from performance advantages that our scheme cannot; nonetheless it is the only other hash-based group signature in the literature. We start the construction with following four parameters: message length s', security parameter s, number of openers o and number of group members N. Each group member owns t secret keys and shares exactly one secret key with each opener. To allow signing an s' bit message with these t secret keys, a W-OTS schemes with parameter w is used. We build a group signature on top of the OTS, by utilizing a transversal design τ-$TD(t,n)$. Note that as the signature scheme remains unchanged, the group signature size is the same as the signature size for a single user scheme. However, group members use different sets of keys which is assigned to them through a transversal design. The design delivers key sets to openers and group members in a related way that satisfies the desired properties of a group signature construction. A $TD(t,n)$ takes $t \times n$ secret keys of size s as input, and delivers a key set of size n to each of the t openers. $TD(t,n)$ also assigns a key set of size t to each of n^τ group members. Note that while the key sets of any two distinct openers have no element in common, the sets given to different group members do. In fact, each secret key appears in $n^{\tau-1}$ different key sets of group members, this is why $t \times n$ total number of keys can provide n^τ group members with t keys each. The final stage of the construction is to connect the group members together by creating the group public key. Our group public verification key consists of t hash pool $HP_1, \cdots, HP_t$ storing the group members' key sets. There are n^τ group members each owning t secret keys. The i^{th} public key of each group member is stored in HP_i. Hence, we get n^τ keys from group members to store in each hash pool. However, these n^τ keys contain only n distinct keys. So, each hash pool stores n elements. With these parameters set, the construction is ready to use.

5.1 Space Complexity

The space complexity in a signature scheme is defined by memory required to store the following three parameters: secret keys (for each user and in total), public verification keys and the signature. In this section, we only consider the space complexity of our primary scheme without the Shamir secret sharing scheme or all-or-nothing transforms applied to the *gok*'s.

a) Private singing key size per user for one message: According to W-OTS each signer needs t secret keys to sign a message.
b) Group private singing key size for one message: Each group user stores t private signing keys, so $t \times N$ keys are stored by group users. However, among these $t \times N$ keys, the number of distinct keys in our scheme is $t \times \sqrt{N}$.
c) Private singing key size per user for B messages: Each user requires a new set of keys per message. Therefore, each user needs to store tB private signing keys.
d) Total private singing key size for B messages: The key sets for different messages are independent of each other, and as we calculated before group members store the total number of $t \times N$ keys to sign one message. Therefore, the group members need to store $t \times B \times N$ private keys in total to sing B messages. Note that, among these keys, $t \times B \times \sqrt{N}$ number of them are distinct keys.
e) Public signing key size for a message: As described in Section 3.3, each user stores all the nodes from the whole Merkle tree, but stores just the authentication path corresponding to its signature in the Go-tree. Therefore, in total there are $2B - 1 + Bt \log \sqrt{N}$ hash values.
f) Verification key: The verifier is only required to have the root of the Merkle tree, i.e., R.
g) Signature size: For signing a message, a group member reveals a set of t keys corresponding to the W-OTS scheme, each accompanied with a path of size $\log \sqrt{N}$ hash values in the Go-tree. The signature of a message also includes the authentication path, $h = \log B$ nodes from the Merkle tree to the root R. Hence, there is a total of $t(1 + \log \sqrt{N}) + \log B$ hash values in a signature.

5.2 Time Complexity

Time complexity is calculated in 4 phases of a signature scheme: Key generation, Signing, and Verification.

a) Key generation and system setup: In order to create the keys for one message, the GK_g generates $t \times n$ nodes for the GO-tree secret keys, divides them into t design groups. Then, a random ordering of numbers from 1 to n is fixed as the shuffling used for that message. Based on this shuffling, the keys are assigned to the t openers, and the N group members through a transversal design, $TD(t, n)$. Then, to create the group public key gpk, t GO-Trees are utilized to implement the hash pools, each storing n hashed secret keys. In the next step, each secret key is assigned to its authentication path in the GO-tree. Finally, the private signing keys, opening keys, and public keys will be distributed among the group members, openers, and publicly, respectively.
   For a multi-signature scheme, we need to repeat the aforementioned steps once for each message to be signed. Therefore, time complexity in the system setup includes generating tn keys, finding a random permutation of n elements, , computing hash functions to obtain hashed parts of the public keys and opening keys, constructing the GO-trees, and distributing the information among members, openers, and publicly, each B times.
b) Signing: According to W-OTS scheme in Section 2.2, signing requires: i) Converting the message M to base-w: $M = (M_1 \cdots, M_{l_1})$, ii) calculating the checksum: $C = \sum_{i=1}^{l_1}(2^w - 1 - M_i)$, iii) converting the checksum to its base w representation: $C = (C_1, \cdots, C_{l_2})$, and iv) computing $\sigma = (\sigma_1, \cdots, \sigma_l) = (\mathcal{H}^{b_1}(sk_1), \cdots, \mathcal{H}^{b_l}(sk_l))$, where $B = (b_1, \cdots, b_l) = M||C$. It also, requires adding the authentication path for the Go-tree and the Merkle tree. However, this information is stored in the memory, it does not impose any further processing time to signing.
c) Verification: In the verification phase, the verifier needs to calculate t hash values for GO-tree roots, each requiring $1 + \log \sqrt{N}$ hash value calculations, totalling to $t(1 + \log \sqrt{N})$. Then $t - 1$ more hash values should be calculated to obtain a leaf of the Merkle tree. Finally, $\log B$ hash values should be computed so that the verifier can compare the final result against the root of the Merkle tree. Therefore, verification requires $\log B + t(2 + \log \sqrt{N}) - 1$ hash calculation.
d) Opening:

5.3 Dealer-User Communication Overhead:

Our scheme provides each user with a copy of the overlaying Merkle tree, which consists of $2B-1$ hash values. For each message, a user receives t private signing keys and their corresponding GO-tree authentication paths, each of length $\log \sqrt{N}$. Hence, each user receives a total of Bt private keys and $B(2 + t \log \sqrt{N}) - 1$ hash values.

5.4 Concrete Parameters and Comparison

To construct a B-time group signature scheme for a group of N members and t openers, we start with the underlying signature scheme. Suppose $B = 2^{20}$ and the messages digests to be signed are of length 256 bits, recall that the hash output size is twice the size of the security parameter, $s' = 2s$, due to collision resistance of the hash functions. Hence, we need a W-OTS scheme with $w = 8$ that generates $t = 35$ private/public keys corresponding Equation 3. According to Theorem 1, we choose a prime $n > t = 35$, i.e., $n = 37$ to supports $N = n^2 = 1369$ number of group members. In this construction, we also consider $\mathcal{H} = \mathcal{H}' =$ SHA-256. Table 1 compares the performance of our scheme against G-Merkle scheme. Table 1 summarizes the comparison

		Parametric		Example	
		GOT	G-Merkle	GOT	G-Merkle
Signature Size		$t(1+\log\sqrt{N})+\log B$	$t+\log B$	$265\times s'$	$55\times s'$
Private Key Size	Per user message	t	t	$35\times s'$	$35\times s'$
	For all messages	tB	tB	$35\times 2^{20}\times s'$	$35\times 2^{20}\times s'$
	Per Group message	tN	tN	$47925\times s'$	$47925\times s'$
	Total	NtB	tB	$47915\times 2^{20}\times s'$	$35\times 2^{20}\times s'$
Average Public Key Size per message		$2+t\log\sqrt{N}$	$\log B+t$	$212\times s'$	$55\times s'$
Maximum Number of signatures per user		B	$\frac{B}{N}$	2^{20}	766

Table 1: Comparing our scheme with G-Merkle for messages of size 128 bits, $N = 1369$ and 2^{20} messages

of the parameters of a group signature construction and the parameters of a single user signature scheme. The security parameter $s = 128$, the message length $s' = 256$ bits, the number of secret keys a single user has $t = 35$ to sign one message. To match the two, number of openers in the group signature is $t = 35$, each holding $n = 37$ secret keys resulting in a group of size $N = 1369$ members.

5.5 Discussion

We exhibited some properties of our scheme in comparison with G-Merkle in Table 1. Our scheme uses same number of secret keys per user per message, t, as G-Merkle does. To enable a group to sign one message, the group members need to store tN private signing keys in total in both schemes. Similarly, to sign B messages, both schemes need the same number of used private signing keys, tB. However, the total number of private signing keys stored in all group members is NtB in our scheme and tB in G-Merkle, since in our scheme each member can sign any of the B messages. In signing time, both schemes perform equally as they both rely on W-OTS scheme to generate the signature, and use a pre-stored path for authentication. Our scheme requires longer signatures and more public keys per message on average than G-Merkle. The disadvantage in some performance evaluations compared to G-Merkle is a result of the fact that our scheme provides more features: *i*) our scheme is flexible in the number of messages each user can sign. A Merkle tree with $B(= 2^h)$ leaves, can support signing any number of messages for group members as long as they add up to B. For G-Merkle however, this structure is pre-defined, since the dealer fixes the nodes' positions for each user and sends them the corresponding keys and authentication paths. *ii*) our scheme utilizes multiple openers to distribute the traceability power among multiple authorities. Note that the openers in our scheme are separate from the key issuer to avoid forming a single point of failure in the scheme, while G-Merkle depends on a single opener who is the key-issuer as well. *iii*) unlike G-Merkle our scheme is salable. Our transition from a one-time to a multi-time signature, does not require any alteration in the deployed Merkle tree. Hence, our scheme can benefit from any advancements that provides Merkle-tree based signatures with flexibility, e.g., as provided by various traversal algorithms [40], or further scalability, e.g., multi-tree [39] approaches.

6 Conclusion

We introduced a construction for a hash-based group signature scheme. Our construction utilizes a hash-based digital signature scheme and turns it to a group signature by adding two layers prior to it. As for the signature scheme that assures unforgeability, we used Winternitz one-time signature (W-OTS). The procedure of adding layers prior to the hash-based signature to turn it to a group signature construction, changes some parameters and preserves some others; The number of the keys a signer requires to sign a message remains unchanged, while the signature size and the number of public keys changes to a greater size to provide a verification tool for all group members. We utilize multiple hash pools in our construction to form the group public key, connect the group members together and support anonymity. On the other hand, the secret keys of the group members are connected to each other via a transversal design. The transversal design takes the set of all group secret keys, and assigns key sets to the group members and the openers in a way that it guarantees traceability for the openers. We provided the proofs for security claims in Section 4 and evaluated the construction in Section 5 using general and concrete parameters, to confirm our solution is a scalable, flexible, group signature scheme that is post-quantum secure as it relies on hash functions and information-theoretically secure structures.

7 Acknowledgement

We would like to thank Douglas Stinson and Douglas Stebila for helpful discussions on the final drafts of the paper.

References

1. Buchmann, J., Dahmen, E., Hulsing, A.: XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In: Post-Quantum Cryptography, LNCS, vol. 7071, pp. 117-129 (2011)
2. Daniel J. Bernstein, Daira Hopwood, Andreas Hlsing, Tanja Lange, Ruben Nieder-hagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, Zooko Wilcox-O'Hearn, SPHINCS: practical stateless hash-based signatures. Date: 2015.02.02. Eu-rocrypt 2015
3. D. McGrew, M. Curcio., S. Fluhrer, Hash-based signatures dreaft-mcgrew-hash-sigs-07, June 20, 2017
4. Lamport, L.: Constructing digital signatures from a one-way function. Technical report, SRI International Computer Science Laboratory (1979)
5. Merkle, R.C., A Certified Digital Signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218238. Springer, Heidelberg (1990)
6. Reyzin, L., Reyzin, N., Better Than BiBa: Short One-Time Signatures with Fast Signing and Verifying". In: Proc. of the 7th Australian Conference on Information Security and Privacy, pp. 144-153. ACISP'02 (2002)
7. Combinatorial Designs: Constructions and Analysis, Douglas R. Stinson, Springer Science & Business Media, 2007
8. Bellare, M., Micciancio, D., Warinschi, B.: Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656. Springer, Heidelberg (2003)
9. Introduction to Modern Cryptography, Jonathan Katz and Yehuda Lindell, Chapman & Hall/CRC, 2007, ISBN:1584885513
10. Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz, Theory and practice of bloom filters for distributed systems, IEEE Communications Surveys and Tutorials 14 (2012), no. 1, 131–155
11. K.M. Martin, M.B. Paterson and D.R. Stinson, Key predistribution for homogeneous wireless sensor networks with group deployment of nodes. ACM Transactions on Sensor Networks 7-2 (2010), article No. 11, 27 pp.
12. Gordon, S.D., Katz, J., Vaikuntanathan, V., A Group Signature Scheme from Lattice Assumptions, In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 395–412. Springer, Heidelberg (2010)
13. D. Chaum and E. van Heyst. Group Signatures. In EUROCRYPT 1991, volume 547 of Lecture Notes in Computer Science, pages 257–265. Springer, 1991.
14. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, CT-RSA 2005, volume 3376 of LNCS, pages 136–153. Springer, February 2005
15. D. Boneh and H. Shacham, Group signatures with verifier-local revocation, In Proceedings of the 11th ACM conference on Computer and communications security (CCS '04). ACM, New York, NY, USA, 168-177, 2004

16. Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, EUROCRYPT 2004, volume 3027 of LNCS, pages 56–73. Springer, May 2004.
17. Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, ACM CCS 04, pages 168–177. ACM Press, October 2004.
18. Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In Serge Vaudenay, editor, EUROCRYPT 2006, volume 4004 of LNCS, pages 427–444. Springer, May / June 2006.
19. G. Ateniese and G. Tsudik. Some open issues and directions in group signature. In Financial Crypto'99, volume 1648 of LNCS, pages 196-211. Springer-Verlag, 1999
20. L. Chen and T. P. Pedersen. New group signature schemes. In A. DeSantis, editor, EURO- CRYPT'94, volume 950 of LNCS, pages 171-181. Springer-Verlag, 1994.
21. Laguillaumie F., Langlois A., Libert B., Stehlé D. (2013) Lattice-Based Group Signatures with Logarithmic Signature Size. In: Sako K., Sarkar P. (eds) Advances in Cryptology - ASIACRYPT 2013. ASIACRYPT 2013. Lecture Notes in Computer Science, vol 8270. Springer, Berlin, Heidelberg
22. Langlois A., Ling S., Nguyen K., Wang H. (2014) Lattice-Based Group Signature Scheme with Verifier-Local Revocation. In: Krawczyk H. (eds) Public-Key Cryptography – PKC 2014. PKC 2014. Lecture Notes in Computer Science, vol 8383. Springer, Berlin, Heidelberg
23. Ling S., Nguyen K., Wang H. (2015) Group Signatures from Lattices: Simpler, Tighter, Shorter, Ring-Based. In: Katz J. (eds) Public-Key Cryptography – PKC 2015. PKC 2015. Lecture Notes in Computer Science, vol 9020. Springer, Berlin, Heidelberg
24. Nguyen P.Q., Zhang J., Zhang Z. (2015) Simpler Efficient Group Signatures from Lattices. In: Katz J. (eds) Public-Key Cryptography – PKC 2015. PKC 2015. Lecture Notes in Computer Science, vol 9020. Springer, Berlin, Heidelberg
25. Martianus Frederic Ezerman, Hyung Tae Lee, San Ling, Khoa Nguyen, and Huaxiong Wang. A provably secure group signature scheme from code-based assumptions. pages 260-285, 2015.
26. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In CRYPTO, pages 41–55, 2004.
27. B. Libert, T. Peters, and M. Yung. Group signatures with almost-for-free revocation. In CRYPTO, pages 571–589, 2012.
28. P. D'Arco, N. N. Esfahani and D. R. Stinson. All or Nothing at All. *Electronic Journal of Combinatorics* **23(4)**, 2016, paper #P4.10, 24 pp.
29. N. N. Esfahani, I. Goldberg and D. R. Stinson. Some Results on the Existence of t-All-or-Nothing Transforms Over Arbitrary Alphabets. *IEEE Transactions on Information Theory*, (**64**)-8, 2018, 3136–3143.
30. B. Libert, T. Peters, and M. Yung. Scalable group signatures with revocation. In EUROCRYPT, pages 609–627, 2012.
31. van Heyst E., Pedersen T. P. How to make efficient fail-stop signatures, In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol.658, pp. 366–377. Springer, Heidelberg (1993)
32. Zaverucha G.M., Stinson D.R., Short one-time signatures, Adv.Math. Commun, 5, 473–488 (2011)
33. N. Courtois, M. Finiasz, and N. Sendrier. How to Achieve a McEliece-based Digital Signature Scheme. In Advances in Cryptology – Asiacrypt'2001, volume 2248 of LNCS, pages 157–174, Gold Coast, Australia, 2001. Springer.
34. Rachid El Bansarkhani and Rafael Misoczki. G-Merkle: A Hash-Based Group Signature Scheme From Standard Assumptions, PQCrypto 2018
35. Wang, Xin and Cui, Jie and Ji, Lijun. Linear (2, p, p)-AONTs exist for all primes p, Designs, Codes and Cryptography, Springer, 1–13, 2019
36. Shamir A. How to share a secret. Communications of the ACM. 1979 Nov 1;22(11):612-3.
37. R.L. Rivest. All-or-nothing encryption and the package transform. *Lecture Notes in Computer Science* **1267** (1997), 210–218 (Fast Software Encryption 1997).
38. D.R. Stinson. Something about all or nothing (transforms). *Designs, Codes and Cryptography* **22** (2001), 133–138.
39. Andreas Hulsing, Lea Rausch, and Johannes Buchmann. Optimal parameters for xmss mt. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos,Edgar Weippl, and Lida Xu, editors,Security Engineering and IntelligenceInformatics: CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regens-burg, Germany, September 2-6, 2013. Proceedings, pages 194–208, Berlin,Heidelberg, 2013. Springer Berlin Heidelberg.
40. Johannes Buchmann, Erik Dahmen, and Michael Schneider. Merkle tree traversal revisited. In Johannes Buchmann and Jintai Ding, editors,Post-Quantum Cryptography, volume 5299 ofLecture Notes in Com-puter Science, pages 63–78. Springer Berlin / Heidelberg, 2008.
41. Jurjen N. E. Bos and David Chaum. Provably unforgeable signatures. In Ernest F.Brickell, editor,Advances in Cryptology—CRYPTO '92, volume 740 ofLecture Notes inComputer Science, pages 1–14. Springer-Verlag, 1993, 16–20 August 1992.
42. E. Sperner. Ein Satz Uber Untermengen einer endliche Menge.Math. Zeit.27(1928) 544–548
43. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press.

44. Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor,CRYPTO, volume 435 ofLNCS, pages218–238. Springer, 1989.
45. Buchmann J., Dahmen E., Ereth S., Hülsing A., Rückert M. (2011) On the Security of the Winternitz One-Time Signature Scheme. In: Nitaj A., Pointcheval D. (eds) Progress in Cryptology – AFRICACRYPT 2011. AFRICACRYPT 2011. Lecture Notes in Computer Science, vol 6737. Springer, Berlin, Heidelberg
46. Hülsing A. (2013) W-OTS+ – Shorter Signatures for Hash-Based Signature Schemes. In: Youssef A., Nitaj A., Hassanien A.E. (eds) Progress in Cryptology – AFRICACRYPT 2013. AFRICACRYPT 2013. Lecture Notes in Computer Science, vol 7918. Springer, Berlin, Heidelberg
47. Buchmann J., Dahmen E., Klintsevich E., Okeya K., Vuillaume C. (2007) Merkle Signatures with Virtually Unlimited Signature Capacity. In: Katz J., Yung M. (eds) Applied Cryptography and Network Security. ACNS 2007. Lecture Notes in Computer Science, vol 4521. Springer, Berlin, Heidelberg