G-Merkle: A Hash-Based Group Signature Scheme From Standard Assumptions

Rachid El Bansarkhani and Rafael Misoczki

Technische Universität Darmstadt, Germany
elbansarkhani@cdc.informatik.tu-darmstadt.de
Intel Corporation, USA
rafael.misoczki@intel.com

Abstract. Hash-based signature schemes are the most promising cryptosystem candidates in a post-quantum world, but offer little structure to enable more sophisticated constructions such as group signatures. Group signatures allow a group member to anonymously sign messages on behalf of the whole group (as needed for anonymous remote attestation). In this work, we introduce G-Merkle, the first (stateful) hash-based group signature scheme. Our proposal relies on minimal assumptions, namely the existence of one-way functions, and offers performance equivalent to the Merkle single-signer setting. The public key size (as small as in the single-signer setting) outperforms all other post-quantum group signatures. Moreover, for N group members issuing at most B signatures each, the size of a hash-based group signature is just as large as a Merkle signature with a tree composed by $N \cdot B$ leaf nodes. This directly translates into fast signing and verification engines. Different from lattice-based counterparts, our construction does not require any random oracle. Note that due to the randomized structure of our Merkle tree, the signature authentication paths are pre-stored or deduced from a public tree, which seems a requirement hard to circumvent. To conclude, we present implementation results to demonstrate the practicality of our proposal.

Keywords: Group Signatures, Hash-Based Signatures, Post-Quantum Cryptography.

1 Introduction

Post-quantum cryptography is attracting increasing attention since the recent announcements by NIST [oSN16], NSA [NSA15] and the PQCRYPTO project [PQC16] that endorse the migration from classical to post-quantum schemes. Hash-based signatures (HBS) are considered good candidates as they offer good security and performance guarantees. They are considered quantum resistant, while widely-deployed public key cryptosystems are susceptible to polynomial-time quantum attacks [Sho94], and rely only on minimal assumptions, namely certain well-studied security notions related to hash functions. Note that any signature scheme (classical or post-quantum) with appended message also relies on the security of hashing (used to map arbitrary length message into a fixed length digest), plus some other (likely less studied) assumptions.

Another strength of HBS refers to their practical performance. As opposed to conventional schemes, where expensive computations are required, HBS only require hash computations, an operation with performance akin to symmetric key cryptography rather than public-key cryptosystems. Given their high efficiency and tight security, HBS can be seen as one of the fewest post-quantum cryptographic alternatives that can immediately replace conventional cryptosystems (although stateful schemes may require extra caution related to state management [MKF$^+$16]). However, the same simplicity that leads to high efficiency and tight security also imposes limitations to build more sophisticated constructions such as group signatures.

Group signatures allow any member of a group to anonymously sign messages on behalf of a group. This is accomplished by a unique group public key that is the same for all group members. A group manager can break the anonymity of any group signature by means of a master key and thus determine the respective issuer (*traceability*). Note that no other entity other than the group manager is able to gather information or trace a signature back to any group member (*anonymity*). Group signature schemes have great applicability in real-world, such as in remote attestation protocols, traffic management, e-commerce, e-cash, e-voting and e-auction, just to name a few examples.

1.1 Related Work

The first group signature scheme has been introduced in [Cv91]. Subsequently, it has been improved in [ACJT00]. The notion of full-anonymity and full-traceability can be traced back to the security model proposed in [BMW03], which allows for even stronger security properties, once these notions are established in a scheme. Since then, a great deal of practical constructions based on classical assumptions have been proposed. Those schemes can be classified into random oracle based constructions [ACJT00, CL02, CL04, DP06] and standard model variants [BMW03, BSZ05, BW06, BW07, Gro07]. All of these constructions are based on Groth-Sahai's non-interactive proof systems (NIZK) for a great deal of languages. In [BL09], Brickell et al. introduced EPID with advanced properties such as signature-based and private-key based revocation. There are only a few (secure) constructions based on computational problems that are believed to be quantum resistant. Such schemes are mainly based on lattice-based hardness assumptions [GKV10, LLLS13, LLNW14, NZZ15, LNW15] or on code-based scheme [ELL$^+$15] that relies on additional (non-usual) assumptions. However, all of those constructions require expensive non-interactive zero-knowledge arguments for specific languages such as [MV03].

Previously, it seemed hard to construct group signature schemes out of hash functions as they offer little structure to exploit. In fact, there exists little literature on special property signature schemes from hash functions. One example is the forward secure signature scheme (also proxy- and key-insulated signature schemes) from one-way functions by Buchmann et al. [BDH11]. Recently, it has also been shown that NIZK proofs [GMO16] can be built out of hash functions and techniques from multi-party computation. This opens new directions as NIZK proof systems serve as a common tool to realize advanced cryptographic constructions. However, to the best of our knowledge, no hash-based group signature scheme has ever been proposed in the literature.

1.2 Our Contributions

In this work, we propose the first (stateful) hash-based group signature scheme. Our proposal has many advantages over other group signature schemes such as:

- It is very simple as it is solely built from a regular Merkle tree based signature scheme in combination with a secure block cipher or pseudo random function. The latter can be built out of one-way functions by the construction of Luby and Rackoff [LR86], hence allowing for a construction based on minimal assumptions in the standard model. This answers the open question raised in [LNW15].
- We do not require expensive non-interactive zero knowledge proofs (e.g. via the Fiat-Shamir Transform) as used in other group signature schemes in order to prove possession of a secret. As a result, no random oracle instantiation is needed.
- It is post-quantum secure with small key and signature sizes, outperforming all other post-quantum group signature alternatives. In fact, the public key size and the underlying one-time signature size are as large as in the single-signer setting. The authentication path increases by $\log N$ nodes as the associated Merkle tree consists of $N \cdot B$ leaf nodes, resulting in the same number of signatures per group member. This coincides with the number of signatures in the single-signer setting.

To realize this functionality we exploit the structure of Merkle trees [Mer90]. More precisely, we let all group members share a very same Merkle tree, which has the leaf nodes shuffled (by means of a block cipher or, more generally, a pseudorandom permutation (PRP)) before the tree is built. Our construction assigns a bounded number B of signatures to each group member. Each group member also has its own secret key. In Section 6, we give several options to handle a limitation related to the authentication paths, providing different trade-offs. We stress that none of these strategies seems to be optimal for all situations, but we hope that this discussion will feed further works in the community on how to optimally address this particular problem.

In terms of efficiency, for N group members, hash function digest size n, the size of our group signature is $[|\text{one-time signature}| + n \cdot (\log N + \log B)]$ bits, which is as large as a Merkle tree signature with $N \cdot B$ leaf nodes. Lattice-based counterparts occupy at least $\log N \cdot \tilde{O}(n)$ bits (ring variant) or $\log N \cdot \tilde{O}(n^2)$ bits (matrix variant) and the group public key size increases by a factor of $\log N$. Additionally, the underlying lattice problem weakens by a factor of $\log N$ (e.g. $\mathsf{SIVP}_{\log N \cdot \tilde{O}(n^2)}$, see [LNW15]). The nature of our construction immediately carries over to the running times for signing and verification. Note that other group signature constructions often rely on costly zero-knowledge proofs (using Fiat-Shamir) to establish the different features of the group signature scheme, whilst our construction inherits them from the Merkle tree structure for free. Our scheme can be instantiated using any (stateful) Merkle-like signature scheme, e.g. the XMSS Merkle tree scheme [BDH11, HBGM18].

In terms of security, we give a proof for full-traceability within the well-known framework of Bellare, Micciancio, and Warinschi (BMW model) [BMW03] and anonymity according to Camenisch and Groth [CG05]. The latter is required for a private key based revocation mechanism, which is a desirable feature and enables the verifier to identify signatures issued by a revoked private key. We also discuss a Proof-of-Concept implementation to show its efficiency and scalability for a publicly available tree.

1.3 Group Signature Scheme by Chaum and van Heyst

Our construction can be seen as an improvement of the very first discrete-log based group signature scheme due to David Chaum and Eugène van Heyst [Cv91]. In their first construction each group member is randomly assigned a number of public and secret keys of a secure signature scheme, where each group member stores its assigned set of secret keys. To open signatures, the group manager stores the group member's name for every issued key. The group public key is represented as the (random) concatenation of all public keys. It can be seen that the approach taken in [Cv91] can be extended to any regular signature scheme. However, in our construction we only have one single hash value as the group public key regardless of the number of signers and signatures. Furthermore, each signer just stores one single secret seed, out of which all one-time key pairs and leaf nodes are derived. To ensure anonymity and traceability at reduced costs, the group manager applies a pseudorandom permutation to shuffle the positions of the leaf nodes and to open signatures without storing a large list of names and respective keys. All these modifications can directly be applied to [Cv91].

1.4 Open Problems

Full-anonymity in the BMW model would lead to a public-key encryption scheme [CG05, AW04] solely based on the existence of one-way functions, i.e. the group signature scheme would also serve as a basis to build other public key cryptographic primitives. However, there is little hope to achieve such a result as the seminal work [IR89] by Impagliazzo und Rudich already showed that one cannot base secure key agreement protocols on one-way functions. Thus, a challenging open problem consists of modifying our scheme in order to satisfy stronger notions of full-anonymity. Obtaining and storing the authentication path, a requirement of Merkle tree constructions, seems a limitation of our work hard to circumvent. Finally, we note that multi-tree approaches (e.g. [HRB13]) are not applicable, thus limiting the maximum attainable tree height.

1.5 Organization

Section 2 presents the preliminary concepts, Section 3 the background on group signature schemes and related security notions and Section 4 presents our construction. Section 5 details its security assessment, Section 6 discusses the authentication path computation, Section 7 its implementation aspects and Section 8 our conclusions.

2 Preliminaries

It is well known [Mer90, BDS08, BDH11] that it is possible to build secure digital signature schemes using only a *secure hash function*. This is an advantage in comparison to any other signing scheme, which require not only a secure hash function but also a hard underlying computational problem. In this sense, hash-based signature schemes achieve minimal security requirements. The concept of secure hash function is vague and requires some refinement. Below we recap three computational problems related to hash functions useful to assess the security of hash-based signature schemes.

A cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^n$ is an efficiently computable function $\mathcal{H} = (\mathsf{H}, \mathsf{HKGen})$, where $\mathsf{HKGen}(1^n)$ outputs a hash function H and H maps on input H and an element $m \in \{0,1\}^*$ to $H(m) \in \{0,1\}^n$. Depending on the given application scenario it may be required to have a certain set of properties [Rog04].

One-wayness (OW) A hash function H is said to be one-way, if it is infeasible for a PPT-adversary to find a preimage m of a random image.

Collision Resistance (CR) A hash function H is said to be collision resistant, if it is infeasible for a PPT-adversary to find two distinct messages $m \neq m'$ that map to the same hash value, i.e., $H(m) = H(m')$.

Second Preimage Resistance (SR) A hash function H is said to be second preimage resistant, if it is infeasible for a PPT-adversary and a given pair $(m, H(m))$ to find another message $m \neq m'$ that maps to the same hash value $H(m) = H(m')$. We note that CR implies SR.

To provide λ bits of classical security against collision and pre-image attacks, a hash function needs to have digest size of at least $n = 2\lambda$ bits and $n = \lambda$ bits, respectively. To maintain the same levels of security in a post-quantum world, the digest of the hash function would need to be extended by a factor $3/2$ (for collision) and by a factor 2 (for pre-image). This is due to speedups induced by Grover's search algorithm [Gro96] on a quantum computer. Note that this quantum speedup is marginal when compared to the one obtained by Shor's algorithm [Sho94] against RSA/ECC cryptosystems.

Most hash-based signature schemes are either one-time (OTS) or multi-time signature (MTS) schemes. OTS schemes (such as Winternitz [Mer90] and W-OTS+ [Hül13]) have an important limitation: a private key must not be used to sign more than one message (if so, it loses its security guarantees). Due to this limitation, Merkle proposed a way to transform an OTS scheme into a MTS scheme solely based on hash functions. This is known as the Merkle tree signature scheme [Mer90] and offers a way to tie many one-time public keys into a single multi-time public key. In this sense, any of the signatures generated by the one-time private keys can be validated with a single (multi-time) public key.

The Merkle scheme uses a binary tree of height h that is built from 2^h one-time key pairs. The leaf nodes are computed as the hash of the one-time public keys. The inner nodes are computed as the hash of concatenated children nodes. This rule is used to build all inner nodes up to the root, which is the multi-time public key.

3 Foundations of Group Signature Schemes

In this section we introduce the different definitions and security notions associated to group signature schemes following the work of Bellare, Micciancio, and Warinschi [BMW03] for full-traceability and anonymity according to Camenisch and Groth [CG05], which describe a comprehensive set of properties.

The appropriate security model for anonymity in our setting is due to [CG05], since the adversary is only given access to the secret keys of corrupt users in contrast to [BMW03], where the adversary has full access to the secret keys of all group members. The former reflects circumstances where the adversary is either static or he is adaptive and the parties cannot erase data (in these cases full-anonymity would not enhance security). In our scheme, we do not achieve full-anonymity in the sense of [BMW03] for an adaptive adversary and parties with erasing capabilities, since revealing the secret keys immediately allow to identify all the associated signatures. But this seems plausible for constructions solely based on the existence of one-way functions. We note that once our hash-based group signature scheme satisfies also full-anonymity following [BMW03] it is possible to build a public-key encryption scheme out of it. This would be a great result in post-quantum cryptography in general as this would imply public key cryptography (see Section 8) solely based on the security of one-way functions (minimal assumptions). In Section 8, we briefly discuss how full-anonymity allows to construct a public key encryption scheme following [CG05, AW04]. Thus, in the security game [CG05] the adversary is given a random signature and he must output the correct identity, under which it has been signed and which has not been corrupted before. We account for the fact that stateful Merkle-like schemes output a bounded number of signatures in the anonymity game.

In a group signature scheme there are essentially 3 parties involved:

- **Group Manager** He instantiates the scheme and generates the group public key. He assigns each group member with a secret key. In case of misuse of group signatures or misbehavior the group manager has the power to reveal the identity of a group signature by means of its master key.
- **Group Member** A group member can sign any data using its secret key such that his identity is concealed from any verifier other than the group manager.
- **Verifier** Any verifier can use the group public key in order to verify a group signature. He only knows that a group member signed the data, but he cannot specify which group member.

The syntax of a group signature scheme and the involved algorithms is as follows.
Syntax: A group signature scheme is composed by the following polynomial-time algorithms $\mathcal{GS} = (\mathsf{G.KGen}, \mathsf{G.Sign}, \mathsf{G.Verify}, \mathsf{G.Open})$.

$\mathsf{G.KGen}(1^k, 1^N)$: The group key generation algorithm is a randomized algorithm that takes as input the security parameter k, the number of users N and generates and outputs a group public key gpk, the group signing keys gsk_i associated to the i-th group member for $i \in [N]$ and the group master key or tracing key gmsk required to open signatures by the group manager.

$\mathsf{G.Sign}(\mathsf{gsk}_i, m)$: The group signing algorithm takes as input a group signing key gsk_i, a message $m \in \{0,1\}^*$ and outputs a group signature σ on the message.

$\mathsf{G.Verify}(\sigma, m, \mathsf{gpk})$: The deterministic group verification algorithm takes as input a group signature, a message and the group public key, and outputs 1 in case the signature is valid, else 0.

$\mathsf{G.Open}(\mathsf{gmsk}, \sigma, m)$: The group opening algorithm is a deterministic algorithm that on input the group master key, a signature, and the corresponding message outputs an identity related to σ.

There are two basic conditions to be satisfied in order for the scheme to work appropriately. In particular, the correctness requirement of the verification and tracing procedure has to be guaranteed for all honestly generated signatures. That is, for any group member $i \in [N]$ the following two expressions have to hold except with negligible probability

$$\mathsf{G.Verify}(\mathsf{G.Sign}(\mathsf{gsk}_i, m), m, \mathsf{gpk}) = 1$$
$$\mathsf{G.Open}(\mathsf{gmsk}, \mathsf{G.Sign}(\mathsf{gsk}_i, m), m) = i\,.$$

The first requirement mainly implies that all honestly generated group signatures must be valid. And the second expression allows the group manager by means of the master key to recover the identity of a correctly generated signature.

We now recap the security notions related to group signature schemes introduced in [BMW03] by Bellare et al. and subsequently relaxed in [BBS04] by Boneh et al. In the relaxed version, the adversary is not permitted to have oracle access to the opening procedure. For anonymity we refer to the security model of Camenisch and Groth [CG05], where the adversary is indeed given access to the secret keys of corrupted group members. This model particularly also captures the possibility to realize private key based revocation, which represents a useful feature in remote attestation protocols as it may be a required feature to identify all signatures of an identity once its secret key gets exposed (e.g. extracted from the TPM) such that a potential adversary is prevented from signing under this identity. Following these models [BMW03, CG05], a group signature scheme is required to ensure two main security features, which we expound below.

3.1 Anonymity

The adversary not in possession of the group master key is not able to unveil the identity of a group member from its group signature. In the respective security game following [CG05] the adversary is given opening access in order to allow the adversary to see the identity of opened signatures. However, these models need to be modified in order take into account the constrained number B of signatures that a group member is able to issue. Therefore, the adversary is allowed to make arbitrary many calls to the opening and signing oracle for corrupted parties (at most B calls). For honest parties the adversary is only allowed to open at most $B-1$ signatures per signer.

We differentiate between SPRP-Anonymity, which strictly follows the anonymity game of [CG05], and PRP-Anonymity, where the adversary is not allowed to have access to the opening oracle. Later, we will show that a pseudorandom permutation such as a secure block cipher can be used to instantiate the scheme satisfying either anonymity notions.

```
Experiment $\mathsf{Exp}^{\mathsf{SPRP},an-b}_{\mathcal{GS},\mathcal{A}}(k,N,B)$
    $(\mathsf{gpk},\mathsf{gsk},\mathsf{gmsk}) \longleftarrow \mathsf{G.KGen}(1^k,1^N)$
    $(\mathsf{st},i_0,i_1,m) \longleftarrow \mathcal{A}^{\mathsf{G.Open}(\mathsf{gmsk},\cdot,\cdot),\mathsf{G.Sign}(\mathsf{gsk}_{\cdot},\cdot),\mathsf{Corrupt}(\cdot)}(\mathsf{choose},\mathsf{gpk})$
    $\sigma \longleftarrow \mathsf{G.Sign}(\mathsf{gsk}_{i_b},m)$
    $j \longleftarrow \mathcal{A}^{\mathsf{G.Open}(\mathsf{gmsk},\cdot,\cdot),\mathsf{G.Sign}(\mathsf{gsk}_{\cdot},\cdot)}(\mathsf{guess},\mathsf{st},\sigma)$
      if $\mathcal{A}$ queried the opening oracle on $m,\sigma$ in the phase guess, return 0
      if $\mathcal{A}$ queried $i_0$ or $i_1$ to $\mathsf{Corrupt}(\cdot)$, return 0
      if the maximal number of queries to $\mathsf{G.Sign}(\mathsf{gsk}_{\cdot},\cdot)$ wrt $i_0$ and $i_1$
        is bounded by $B-1$, return $j$ else $\perp$.
```

Fig. 1. Experiment for SPRP-anonymity

Definition 1 (SPRP-Anonymity [CG05]). *Formally, a group signature scheme defined by the algorithms* $\mathcal{GS} = (\mathsf{G.KGen},\mathsf{G.Sign},\mathsf{G.Verify},\mathsf{G.Open})$ *is called anonymous, if for all probabilistic polynomial adversaries* $\mathcal{A}$ *with access to the opening and signing oracles and all polynomially bounded* N *the advantage of the adversary in the experiment* $\mathsf{Exp}^{\mathsf{SPRP},an}_{\mathcal{GS},\mathcal{A}}(k,N,B)$ *is negligible*

$$\mathsf{Adv}^{\mathsf{SPRP},an-b}_{\mathcal{GS},\mathcal{A}}(k,N,B) = \left|P[\mathsf{Exp}^{\mathsf{SPRP},an-1}_{\mathcal{GS},\mathcal{A}}(k,N,B)=1] - P[\mathsf{Exp}^{\mathsf{SPRP},an-0}_{\mathcal{GS},\mathcal{A}}(k,N,B)=1]\right|.$$

We also define a weaker form of anonymity taking into account the relaxation considered in [BBS04]. In particular, the adversary is not granted access to the opening oracle in the experiment. As we will see later, this will allow us to instantiate the scheme with only a secure block cipher.

Definition 2 (PRP-Anonymity). *A group signature scheme defined by the algorithms* $\mathcal{GS} = (\mathsf{G.KGen},\mathsf{G.Sign},\mathsf{G.Verify},\mathsf{G.Open})$ *is called anonymous, if for all probabilistic polynomial adversaries* $\mathcal{A}$ *with access to signing oracle and all polynomially bounded* N *the advantage of the adversary in* $\mathsf{Exp}^{\mathsf{PRP},an}_{\mathcal{GS},\mathcal{A}}(k,N,B)$ *is negligible*

$$\mathsf{Adv}^{\mathsf{PRP},an-b}_{\mathcal{GS},\mathcal{A}}(k,N,B) = \left|P[\mathsf{Exp}^{\mathsf{PRP},an-1}_{\mathcal{GS},\mathcal{A}}(k,N,B)=1] - P[\mathsf{Exp}^{\mathsf{PRP},an-0}_{\mathcal{GS},\mathcal{A}}(k,N,B)=1]\right|.$$

```
**Experiment** $\mathsf{Exp}^{\mathsf{PRP},an-b}_{\mathcal{GS},\mathcal{A}}(k,N,B)$
    $(\mathsf{gpk},\mathsf{gsk},\mathsf{gmsk}) \longleftarrow \mathsf{G.KGen}(1^k,1^N)$
    $(\mathsf{st},i_0,i_1,m) \longleftarrow \mathcal{A}^{\mathsf{G.Sign}(\mathsf{gsk}_{\cdot},\cdot),\mathsf{Corrupt}(\cdot)}(\mathsf{choose},\mathsf{gpk})$
    $\sigma \longleftarrow \mathsf{G.Sign}(\mathsf{gsk}_{i_b},m)$
    $j \longleftarrow \mathcal{A}^{\mathsf{G.Sign}(\mathsf{gsk}_{\cdot},\cdot)}(\mathsf{guess},\mathsf{st},\sigma)$
        if $\mathcal{A}$ queried $i_0$ or $i_1$ to $\mathsf{Corrupt}(\cdot)$, return 0
        if the maximal number of queries to $\mathsf{G.Sign}(\mathsf{gsk}_{\cdot},\cdot)$ wrt $i_0$ and $i_1$
          is bounded by $B-1$, return $j$ else $\perp$.
```

Fig. 2. Experiment for PRP-anonymity

3.2 Full-Traceability

This feature allows the group manager or possessor of the master key to revoke the anonymity of a signer and unveil its identity. Such a mechanism is important if misbehavior or misuse of the private key has been detected. In fact, this notion is even stronger as it is required that any set of colluding parties should not be able to create signatures that cannot be opened by the group manager or traced back to a group member, even if the parties have access to the master key, for instance extracted during key generation. We note that in this model we do not require to put a bound on the number of exchanged signatures.

Definition 3 (Full-Traceability [BMW03]). *Formally, the group signature scheme* $\mathcal{GS} = (\mathsf{G.KGen},\mathsf{G.Sign},\mathsf{G.Verify},\mathsf{G.Open})$ *is called fully traceable, if for all probabilistic polynomial adversaries $\mathcal{A}$ with access to the opening oracle and all polynomially bounded N the advantage of the adversary in the experiment $\mathsf{Exp}^{f-trace}_{\mathcal{GS},\mathcal{A}}(k,N)$ is negligible*

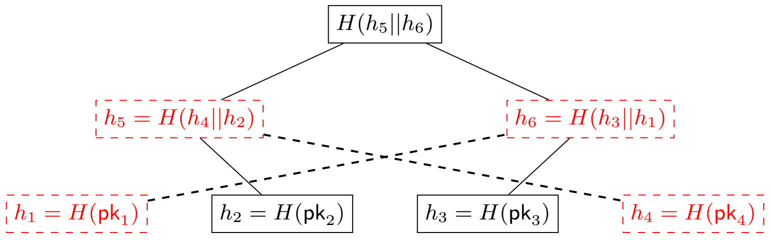
$$\mathsf{Adv}^{f-trace}_{\mathcal{GS},\mathcal{A}}(k,N) = P[\mathsf{Exp}^{f-trace}_{\mathcal{GS},\mathcal{A}}(k,N)=1]\,.$$

```
**Experiment** $\mathsf{Exp}^{f-trace}_{\mathcal{GS},\mathcal{A}}(k,N)$
    $(\mathsf{gpk},\mathsf{gsk},\mathsf{gmsk}) \longleftarrow \mathsf{G.KGen}(1^k,1^N)$
    $\mathsf{st} \longleftarrow (\mathsf{gpk},\mathsf{gmsk})$
    $(m,\sigma) \leftarrow \mathcal{A}^{\mathsf{G.Sign}(\mathsf{gsk}_{\cdot},\cdot),\mathsf{Corrupt}(\cdot)}(\mathsf{guess},\mathsf{st})$
        if $\mathsf{G.Verify}(\sigma,m,\mathsf{gpk}) = 0$, return 0
        if $i$ was queried to $\mathsf{Corrupt}(\cdot)$, return 0
        if $(i,m)$ was queried to $\mathsf{G.Sign}(\mathsf{gsk}_{\cdot},\cdot)$, return 0
        if $\mathsf{G.Open}(\mathsf{gmsk},\sigma,m) = \perp \vee \exists\, i \in [N]$ with $\mathsf{G.Open}(\mathsf{gmsk},\sigma,m) = i$
            return 1
```

Fig. 3. Experiment for full-traceability

4 G-Merkle: A Hash-Based Group Signature Scheme

Our stateful group signature scheme is based on the usage of a Merkle tree, as used in single-signer hash-based signature schemes. The core idea is to extend this approach to a multi-user setting, where more than one signer share the same tree in order to sign messages. A first attempt towards this direction consists in letting each user generate its own Merkle tree (as in the single-signer scheme). Then, each of those sub-trees could be appended to a super tree that will have as leaf nodes the root nodes of the sub-trees.



Hash-based Merkle tree in a multi signer setting, where the nodes are mixed. The nodes $\{\mathsf{pk}_1, \mathsf{pk}_2\}$ belong to Signer 1 and the nodes $\{pk_3, pk_4\}$ belong to Signer 2.

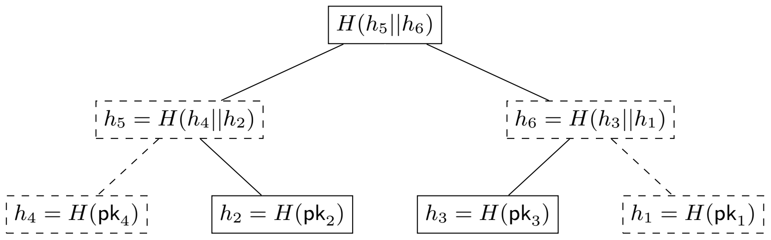


Fig. 4. Hash-based Merkle tree in a multi signer setting after shuffling the nodes. The nodes $\{\mathsf{pk}_1, \mathsf{pk}_2\}$ belong to Signer 1 and the nodes $\{\mathsf{pk}_3, \mathsf{pk}_4\}$ belong to Signer 2.

This naïve construction however does not meet the unlinkability property, a requirement of group signature schemes, since the authentication paths of all signatures issued by a certain signer would share at least one node (the root of its sub-tree). In order to overcome this obstacle, we apply a mixing strategy to the leaf nodes prior to the tree construction. That is, in the key generation phase, the group manager takes the set of leaf nodes from all parties and subsequently applies a secure uniform random permutation to the sets. That is, the permutation will mix the indices and hence the positions associated to the leaf nodes in the combined set. Subsequently, the super-tree is built as described before. In fact, this is a generic way of instantiating hash-based group signature schemes, which can be considered as an extension of the single-signer setting. Since our mixing strategy is based on the usage of pseudorandom permutations, we start with some required definitions. Let $\mathcal{P}_n$ denote the set of permutations and $P \in \mathcal{P}_n$ with $P : \{0,1\}^n \longrightarrow \{0,1\}^n$.

Definition 4. *A pair of functions (E, D) with $E, D : \{0,1\}^k \times \{0,1\}^n \longrightarrow \{0,1\}^n$ is called (t, ϵ) pseudo random permutation, if E_r and D_r are inverses of each other for every $r \in \{0,1\}^k$ and for any probabilistic polynomial adversary the success probability to distinguish a pseudorandom permutation from a truly random premutation is given by*

$$\mathsf{Adv}_{\mathcal{A}}^{Dist}(k, N) = |P_K[\mathcal{A}^{E_K, D_K}(\cdot) = 1] - P_{P \in \mathcal{P}_n}[\mathcal{A}^{P, P^{-1}}(\cdot) = 1]| \leq \epsilon$$

4.1 Instantiating PRPs from One-Way Functions

We note that while it is possible to instantiate pseudorandom permutations by use of block ciphers, pseudorandom permutations can particularly also be built from pseudorandom functions. More precisely, Luby and Rackoff [LR86] propose PRPs using pseudorandom functions combined with the Feistel construction. Goldreich, Goldwasser and Micali [GGM86] showed that pseudorandom generators imply pseudorandom functions, which in turn can be derived from any one-way function [HILL93]. This shows, that one-way functions indeed suffice to construct secure PRPs. Below we give a very simple way to generate pseudorandom permutations from pseudorandom functions.

Theorem 1 (Theorem 3.1,[NR96]). *Let f_1, f_2 be independent pseudo random functions of length n and p_1, p_2 independent permutations of length $2n$. Define the functions*

$$PRP(p_1, f_1, f_1) = T_{f_1} \circ T_{f_2} \circ p_1$$
$$SPRP(p_1, p_2, f_1, f_1) = p_2^{-1} \circ T_{f_1} \circ T_{f_2} \circ p_1,$$

where $T_{f_i}(l, r) = (r, l \oplus f_i(r))$ for $|l| = |r| = n$ and f_1, f_2, p_1, p_2 are chosen independently. Then, $PRP(p_1, f_1, f_1)$ is a pseudorandom function and $SPRP(p_1, p_2, f_1, f_1)$ is a strong pseudo random permutation.

By use of a suitable pseudorandom function, we can instantiate the group signature scheme with a secure SPRP with the aid of Theorem 1. In this case, the whole group signature scheme is just based on the existence of one-way functions, the minimal requirement for the existence of public key cryptography.

4.2 Instantiating PRPs from Block Ciphers

In general, one could apply a perfect uniform random permutation, where a permutation is chosen uniformly at random from a set of $n!$ elements. However, selecting an element from such a huge set requires at least $O(n \log(n))$ bits. For $n > 2^{20}$, this approach is impractical.

Thus, in practice, it is more efficient to instantiate pseudorandom permutations by means of block ciphers such as AES, SIMON and many others that satisfy the conditions from Definition 4. More specifically, the functions $E_K(\cdot)$ and $D_K(\cdot)$ correspond to the encryption and decryption functions of the respective block ciphers. Block ciphers represent a subset of all possible permutations.

Security of Block Cipher. When instantiating the scheme with a secure block cipher it is essential for anonymity that an adversary seeing a number T of (leaf position, group signer)-pairs ($T > 0$ for SPRP-anonymity and $T = 0$ for PRP-anonymity) cannot correctly map leaf positions to group members for the remaining leaves with non-negligible advantage. For instance, if a permutation is sampled from the set of all possible permutations, each group member may be associated to a remaining leaf supposing he did not issue all its signatures. In practice, this means that either a permutation is chosen uniformly at random from the set of all permutations (e.g. for a tree with 4 nodes) or the bit security of the block cipher is larger than or equal to the target security level of the scheme.

Block Ciphers with Larger Output Sizes. In practice, one does not find block ciphers permuting 10-bit or 20-bit integers (as it would be needed for $h = 10$ or $h = 20$) with security more than 100 bits. In this case, one can use larger block ciphers such as AES-128 or AES-256. The tree is then built slightly different. Once the manager receives all leaf nodes, it generates the set of tuples $S = \{(\mathsf{leaf}_1, E_K(1)), \ldots, (\mathsf{leaf}_{2^h}, E_K(2^h))\}$ which contains leaves and the associated encrypted positions (128-bit or 256-bit integers, which are larger than the number of leaves 2^h). For instance, $(\mathsf{leaf}_1, \ldots, \mathsf{leaf}_B)$ belong to group member 1 and $(\mathsf{leaf}_{B+1}, \ldots, \mathsf{leaf}_{2B})$ to group member 2 and so on. Subsequently, the manager sorts the elements of S in an increasing order with respect to the second component. The new order represents the new positions of the leaves in the shuffled tree. The first layer of nodes is then built by not only including the leaves in the hashes but also the encrypted values of the respective leaves, e.g. $h_{i,j} = \mathsf{H}(\mathsf{leaf}_i, E_K(i) || \mathsf{leaf}_j, E_K(j))$. All other tree layers up to the root are built as usual without any further modification. Due to this change, the encrypted indices are part of the authentication path and the group manager can thus open signatures in case of misbehavior. For security, an adversary only sees 2^h encrypted indices somehow mapped to the initial positions (does not even know the index-ciphertext pairs). In the worst-case, the security of the block cipher will not decrease by more than $\log 2^h$ bits.

4.3 Our Construction: (Stateful) G-Merkle

In this section, we employ our group signature scheme on any Merkle tree based signature scheme. However, we keep our construction as general as possible by not restricting to any specific one-time signature scheme. In what follows, let $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ denote the set of algorithms applied in a regular Merkle tree signature scheme.

$\mathsf{G.KGen}(1^k, 1^N)$: The group manager generates the master key $\mathsf{gmsk} \in \{0,1\}^k$ and initializes a block cipher $(E_{\mathsf{gmsk}}(\cdot), D_{\mathsf{gmsk}}(\cdot))$. Each user $i \in [N]$ is assigned a random secret key gsk_i associated to a secure one-time signature scheme such as Winternitz for B (e.g. $B = 2^t$) leaf nodes. That is, the group manager invokes N times $\mathsf{G.KGen}$ as in a regular Merkle tree signature scheme, however outputting only the secret key gsk_i and the hashed public keys serving as leaf nodes. The group manager proceeds as follows:

1. The set of indices associated to the leaf nodes of all users is shuffled
$$\mathsf{Shuffle}(1, \ldots, N \cdot B) = (j_1, \ldots, j_{N \cdot B})$$
where $j_s = E_{\mathsf{gmsk}}(s)$ for $s \in [N \cdot B]$. For instance, leaf node 1 is placed into postion j_1 in the tree (see Figure 4).
2. Subsequently, the group manager builds the G-Merkle tree on top of the shuffled set of nodes and generates the group public key gpk, which is represented as the root node of the G-Merkle tree.
3. Finally, the group manager transfers to user i the set of permuted indices
$$S_i = \{j_{(i-1)B+1}, \ldots, j_{i \cdot B}\}$$
associated to the user's leaf nodes. As for the authentication path, the group manager can choose from several options to compute the authentication path for a signature. We refer to Section 6 for an overview.

$\mathsf{G.Sign}(\mathsf{gsk}_i, m)$: User i maintains a counter t and a list of tuples

$$\mathsf{state} = \{((i-1)B + 1, E_{\mathsf{gmsk}}((i-1)B + 1), \ldots, (i \cdot B, E_{\mathsf{gmsk}}(i \cdot B))\}$$

defining the possible states in the signing process. Whenever the user wishes to sign a message, he takes the actual state

$$\mathsf{state}[t] = [(i-1)B + t, E_{\mathsf{gmsk}}((i-1)B + t)]$$

from the list and sets $t := t + 1$, where the first component serves to internally identify the node with its associated secret key and the second component defines the position of the leaf node within the G-Merkle tree in order to deduce the appropriate authentication path. Supposing that the j-th leaf has been used to sign, following the authentication path consists of the nodes $(a_0, \ldots, a_{h-1})$, where h is the tree height. Define $\ell := \lfloor j/2^h \rfloor$ and denote by $v_j[k]$ the k-th node at the j-th layer, then we have

$$a_j = \begin{cases} v_j[\ell - 1] \text{ for } \ell \equiv 1 \bmod 2 \\ v_j[\ell + 1] \text{ for } \ell \equiv 0 \bmod 2\,. \end{cases}$$

Finally, by use of the secret key gsk_i the signing algorithm outputs a group signature (σ, m) on a message m that is composed by a one-time signature produced by the underlying signing algorithm Sign and the authentication path (see Section 6).

$\mathsf{G.Verify}(\sigma, m, \mathsf{gpk})$: The deterministic group verification algorithm invokes Verify of the underlying signature scheme on the G-Merkle tree taking as input a group signature σ, a message m and the root node of the G-Merkle tree.

$\mathsf{G.Open}(\mathsf{gmsk}, \sigma, m)$: On input the signature containing the exact authentication path, which can be represented by the position of the leaf node and the intermediate nodes, the manager extracts the position l and invokes the decryption algorithm $D_{\mathsf{gmsk}}(l) = j$ for $l \in [N \cdot B]$, otherwise he outputs $\perp$ and aborts. Subsequently, he identifies the set S_i with $|S_i| = B$ s.th. $j \in S_i$ and outputs i.

Possible Modifications. We note that in practice one rarely finds PRPs with small output sizes. In this case, one applies our simple modifications from Section 4.2 using any secure blockcipher. We further note that one may adopt a dynamic approach in the key generation phase, where the group members individually generate their secret keys and associated leaf nodes. Subsequently, the leaf nodes are handed over to the group manager who shuffles the leaves and builds the Merkle tree on top of this mixed set. Such a strategy prevents the group manager from knowing the secrets and the risk of attacking the secrets is minimized.

5 Security

In this section, we prove that the construction proposed in Section 4 provides anonymity following [CG05] and full-traceability according to [BMW03]. Unforgeability of the scheme follows directly from the underlying signature scheme.

5.1 Anonymity

For anonymity, we prove the variant of Definition 1 where the adversary is allowed to have access to the secret keys of corrupt group members. The adversary must guess under which honest identity the signature has been created. Depending on the circumstance that the adversary is given or refused access to the opening oracle in the choose stage, we require the underlying block cipher to be an SPRP or PRP.

Theorem 2. *Let $N \cdot B$ be the number of leaves in the G-Merkle tree and N the number of users. Then, the construction described in Section 4.3 provides SPRP-anonymity following the experiment in Definition 1 under the assumption that a strong pseudorandom permutation is employed to shuffle the leaf nodes.*

Proof. The proof of this theorem is very simple as the adversary represents now an SPRP adversary involved in an indistinguishability game, however not having access to the encryption oracle. That is, assume there exists an adversary that breaks the anonymity of the scheme, then we can build an algorithm that distinguishes the encryption of different plaintexts derived under a strong pseudorandom permutation. In particular, the adversary is initially given access to the signing oracle and the opening oracle, which essentially represents the decryption algorithm, on group signatures of its choice during the choose stage. Due to the constrained tree size, the adversary can query the signing (and hence opening oracle on different signatures) at most $B-1$ times for each honest member such that each user still has the chance to generate one last signature. He can also corrupt arbitrary group members. Eventually, the adversary outputs some state information st, an arbitrary message m to be signed and two existing identities i_0, i_1, that have not been corrupted. In the second stage, the adversary gets as input the state information st and a signature σ on m under an identity, which is selected uniformly at random from i_0 and i_1. The adversary is now challenged to make a right guess on the identity used to sign the message m. In this stage, the adversary still has access to the signing and opening oracle on signatures other than σ. The condition that each user can still sign a message prevents the adversary to exhaust all leaf nodes and make a trivial guess via exclusion based on the remaining signature not queried to the opening oracle yet.
The only element of a group signature that depends on the identities is the position of a leaf in the G-Merkle tree. All other elements can be replaced by the corresponding distributions that are independent from the identities. According to our construction from Section 4.3, the index set of the leaf nodes owned by signer S_k is $\{k_0, \ldots, k_{B-1}\} = E_{\mathsf{gmsk}}(\{(k-1) \cdot B, \ldots, k \cdot B - 1\})$. We can safely assume that all but 1 plaintext-ciphertext pair per honest identity have been revealed (each signer can still sign one last message) such that the remaining plaintexts $p_0 \in [N \cdot B]$ and $p_1 \in [N \cdot B]$ of i_0 and i_1 are known to the adversary. The output of the challenger is a random ciphertext c that either encrypts p_0 or p_1. Under the SPRP assumption of the cipher, we can replace the ciphertext set, in particular c as well, by random elements independent from the plaintext or indices related to an identity. As a result, the claim follows. We note that if a perfect permutation was used to encrypt the indices, then the probability to have a particular plaintext is equal for any ciphertext given by the challenger. $\square$

In case the adversary is not granted access to the opening oracle in the experiment from Definition 2, we can even prove PRP-anonymity with only the requirement of using a pseudorandom permutation (analogously to Theorem 2).

Theorem 3. *Let $N \cdot B$ be the number of leaves in the G-Merkle tree and N the number of users. Then, the construction described in Section 4.3 provides PRP-anonymity in accordance to the experiment in Definition 2 assuming a pseudorandom permutation is employed to shuffle the leaf nodes.*

5.2 Traceability

Full-traceability subsumes a collection of other properties as described in [BMW03]. Following the traceability experiment given in Figure 3, the proof relies on the unforgeability of the underlying signature scheme. In fact, our G-Merkle scheme inherits its existential unforgeability immediately from the basic scheme as described in Section 2. In general, if the basic Merkle tree construction is secure against such an adversary, then so is the G-Merkle tree construction. Thus, we call our group signature scheme G-Merkle$_{XMSS}$ if it relies on XMSS.

Theorem 4. *Let T be a hash-based one-time/few-time signature scheme and Merkle$_T$ the corresponding Merkle tree based multi-time signature scheme. If Merkle$_T$ is existentially unforgeable under chosen message attacks, then so is the group signature scheme G-Merkle$_T$.*

The proof of this statement is straightforward as the multisigner G-Merkle$_T$ scheme differs from a single-signer Merkle$_T$ scheme in the shuffling procedure, where the order of the nodes is changed, and the number of participants with their own secret keys.

Theorem 5. *Let T be the number of leaves in G-Merkle as defined in Section 4. Suppose that there exists a* PPT *traceability adversary, then there exists an algorithm $\mathcal{B}$ that breaks the unforgeability of the underlying signature scheme.*

Proof. The proof of this theorem is mainly based on Theorem 4 stating the existential unforgeability of the underlying digital signature scheme. We show that such an adversary does not exist unless the underlying signature scheme is insecure. We proceed by means of the experiment defined in Figure 3 following [BMW03].

Suppose, there exists an attacker that can successfully generate such a forgery. Clearly, in the G-Merkle tree construction the height of the tree $\log_2(N \cdot B)$ and number of leaves $N \cdot B$ is known in advance such that a group signature can only be valid if the index of the leaf node used to sign the message is an element of $\{0, \ldots, N \cdot B\}$. Therefore, the adversary must produce a forgery that opens to an identity of an honest user $i \notin \mathcal{C}$, which correctly verifies. However, this is only the case if the attacker breaks the unforgeability of the underlying signature scheme (Merkle tree construction), since he is not in possession of the secret key associated to the identity i. Due to Theorem 4 such an adversary does not exist. □

6 Authentication Path Computation

The G-Merkle tree is composed by leaf nodes originating from different users. Thus, the conventional approach of generating the authentication path is not immediately applicable as the authentication path inherently requires the knowledge of the other nodes. As a result, we need a different strategy in order to derive the authentication path. In what follows, we propose some possible solutions to tackle this target.

Public Leaf Nodes. The first approach works similar to the very first Merkle tree constructions, where a user stores each leaf node of the associated tree. Translating this strategy to the multi-user setting the group manager publishes all leaf nodes of all users. This leads to a storage size of at most $N \cdot B$ nodes. Whenever a user invokes its signing algorithm it combines the leaf nodes in order to generate the authentication path associated to its one-time signature. Alternatively, the whole tree can be published/stored and the memory requirement just double. In this case the running times for signing decrease as group members are no longer required to compute the inner nodes.

User directed Authentication Path Computation. The group manager can also send to each user the authentication paths together with the associated indices during key generation. The user stores the nodes and can delete those nodes once they are consumed.

Lemma 1. *Let N denote the number of group members and B the number of potential signatures per user. Then the memory size* Mem *of a user is bounded by*

$$\log N \leq \mathsf{Mem} \leq B \cdot \log(N) + B < N \cdot B$$

Proof. The best possible case occurs when all nodes of a user are neighbors such that he can generate many of the entries in the authentication path by use of his key pair. However, in this case, he can build a subtree of height $1 + \log_2 B$ and requires to store $\log N$ nodes in order to build the authentication path. In the worst-case the user stores the nodes of the $\log B$-th layer, i.e. B nodes, which allow him to generate all other nodes in the upper part of the G-Merkle tree. This is due to the fact that all B potential signatures have to cross one of the B nodes in the $\log B$-th layer. Furthermore, he has to store at most $B \cdot \log N$ nodes from the remaining $\log N$ layers of the tree, which corresponds to the nodes in the authentication path. $\square$

In practice each user can determine the exact number of nodes to be stored and hence optimize the memory size. He can eliminate duplicated nodes that appear in multiple authentication paths. For instance, all authentication paths have to use either the left or right child of the root node. The user is doing best if he chooses to store both right and left children (if req.) only once.

Improved Storage Size with Clustering. Based on the proof of Lemma 1 and the observation that the memory sizes improve, if the leaf nodes of a group member are close to each other, it is possible to split the group into several clusters. This is accomplished secretly by the group manager. The leaf nodes of the G-Merkle tree are then clustered accordingly. For instance, we partition the leaf nodes into k clusters, where each cluster contains all nodes of N/k users. This enhances the probability for each group member to use many of the nodes in the authentication path several times such that the absolute storage requirement is reduced. Due to the fact that the verifiers and group members themselves are not aware of how the leaf nodes are partitioned, how many clusters do exist and who are the group members within a particular cluster, the anonymity is still guaranteed in case the adversary is not given access to the opening oracle within the anonymity game. Such a clustering strategy is advantageous if N is very large. A further advantage of the clustering strategy is the usage of block ciphers of small output sizes. In fact, the output size can be chosen equal to the cluster size.

In case the adversary is given access to the opening oracle, the adversary at most learns which users belong to a cluster. This reduces anonymity in the whole set of group members to anonymity of the smaller cluster. But the adversary is not able to map a signature to a certain group member of a cluster in the challenge phase. Assuming that $N = 2^{t_1}$ and the cluster size is $k = 2^{t_2}$, the authentication path of users from a same cluster have always the same last $\log(N \cdot B) - \log(N \cdot B/k) = t_2$ nodes in the authentication path. This is due to the fact that a certain cluster has the same parent node at height $\log(N \cdot B/k)$. From then on, the sibling nodes are identical.

Interactive Authentication Path Computation. In case the group manager maintains a list of the group signers secret keys, it is possible to ask the group manager the required authentication path in an online fashion. Once a group member sends its part of the signature (OTS) together with the leaf position to the verifier, the verifier can invoke the group manager for the associated authentication path. Clearly, this has no impact on the security as all leaf nodes can be made public.

7 Implementation

In this section, we discuss the implementation aspects of our proposal. This discussion is based on our G-Merkle implementation in C, which is an extension of a XMSS/WOTS+ implementation as specified in [HBGM18], using SHA2-256 as the hash function.

Our implementation follows the approach described in Section 4.2 where block ciphers with larger outputs are used. More precisely, we use AES-256 to perform the indices encryption (which acts as the shuffling process of the leaf nodes). Therefore, the leaf nodes indices are initially represented as 256-bits long integer numbers (padded with zeros on the left) and then encrypted using AES-256. The 256-bits ciphertexts are considered as the new (shuffled) leaf node indices. Note that most of the encrypted indices will likely be out of the range $[0, 2^h - 1]$, but (as described in Section 4.2) G-Merkle only cares about the ordering in which these encrypted indices appear. These encrypted indices are also used to open signatures. Once the encrypted indices are computed, they are sorted in increasing order. Our implementation uses a simple Quicksort implementation for very large (256-bits long) numbers. We note that speedups in this step might be achieved by using other sorting algorithms (e.g. Radix sort).

Table 1 shows the performance data of the G-Merkle inner processes. We fix $N = 64$ as the number of group members (also called users) to facilitate the comparison of different tree heights (but other values are possible, given the trade-off between group members and number of signatures). The performance data are given in thousands of cycles measured in an Intel(R) Core(TM) i5-63000 CPU @2.40GHz with 16 GB of RAM. The code has been compiled with GCC 6.4.0 with -O3 compilation flag. Each process has been repeated 100 times and the number of cycles averaged. The first column represents the relevant processes in G-Merkle, namely the generation of the leaf nodes (each user has to perform this step, i.e. generate all OTS key pairs and the corresponding leaf nodes), the encryption and sorting of all indices, the Merkle tree building process, XMSS signature generation, XMSS signature verification and signature opening (which consists of a call to AES-256 decryption). Between parentheses, we denote whether the operation is performed by each user (U) or only by the group manager (GM). The most expensive operation consists of building the tree, an operation handled by the group manager only, not impacting the users. The actual group management operations, such as encryption and sorting of the leaf node indices, and opening of signatures, do not represent any significant overhead, while the XMSS algorithms (sign and verify) are efficient. Our implementation assumes that the Merkle tree is publicly (and securely) available, as discussed in Section 6, thus the authentication path computation (a somewhat expensive operation in XMSS) is not relevant here.

8 Conclusion and Discussion

We introduced the first (stateful) hash-based group signature scheme and showed that it is based on standard assumptions and not on expensive non-interactive zero knowledge proof systems, as seen in other group signature schemes. Our approach exploits

Process (Owner)	$N = 64$		
	$(h = 14,\ B = 256)$	$(h = 16,\ B = 1024)$	$(h = 18,\ B = 4096)$
Generate leaf nodes (U)	$2,319,508$	$9,302,171$	$35,646,646$
Encrypt indices (GM)	$56,960$	$225,818$	$934,001$
Sorting (GM)	$16,866$	$85,767$	$364,334$
XMSS tree building (GM)	$24,347,871$	$114,011,307$	$440,567,352$
XMSS sign (U)	$7,052$	$7,153$	$7,059$
XMSS verify (U)	$9,007$	$9,092$	$9,398$
Signature opening (GM)	100	99	102

Table 1. G-Merkle Performance (in kcycles). U = User, GM = Group Manager.

the structure of Merkle trees in general. Due to this fact, we can generate group signatures more efficiently in terms of running times, signature and key sizes. It is worth mentioning that the provisioning of the authentication paths is challenging. By merging different trees from different users and randomizing the tree structure, the simplicity to generate authentication paths as in XMSS is lost at the benefit of anonymity. We presented several ways to obtain the authentication path (such as publishing the whole Merkle tree), but stress that it is still an open problem how to address it optimally. Furthermore, we strongly emphasize that full-anonymity would result in a secure public key encryption scheme. To illustrate the transformation of a group signature scheme into an encryption scheme, once full-anonymity is achieved, the group manager generates the master key and all secret keys of the different “identities”. The group manager publishes the secret keys as the public key and keeps the master key secret. If a party wishes to encrypt a message, he encodes this message in terms of an identity (or identities) and sends the signature to the group manager, who in turn decrypts the ciphertext by opening the signature. He reveals the identity, which represents the encoded message. Due to full-anonymity an adversary cannot unveil the identity given all secret keys of the identities. Such a result would have a huge impact in cryptography in general as it would allow to build public key cryptography solely based on the existence of one-way functions. This however would somehow oppose the results of [IR89], which states that one-way functions are not sufficient to build key agreement protocols.

8.1 Acknowledgements

We thank Andreas Hülsing for engaging in helpful discussions and the anonymous reviewers for providing detailed comments.

References

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany.

[AW04] Michel Abdalla and Bogdan Warinschi. On the minimal assumptions of group signature schemes. In Javier López, Sihan Qing, and Eiji Okamoto, editors, *ICICS 04: 6th International Conference on Information and Communication Security*, volume 3269 of *Lecture Notes in Computer Science*, pages 1–13, Malaga, Spain, October 27–29, 2004. Springer, Heidelberg, Germany.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss - a practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 – December 2, 2011. Proceedings*, pages 117–129. Springer, 2011.

[BDS08] Johannes A. Buchmann, Erik Dahmen, and Michael Schneider. Merkle tree traversal revisited. In *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008, Cincinnati, OH, USA, October 17-19, 2008, Proceedings*, pages 63–78, 2008.

[BL09] Ernie Brickell and Jiangtao Li. Enhanced privacy ID from bilinear pairing, 2009.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153, San Francisco, CA, USA, February 14–18, 2005. Springer, Heidelberg, Germany.

[BW06] Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 427–444, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[BW07] Xavier Boyen and Brent Waters. Full-domain subgroup hiding and constant-size group signatures. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 1–15, Beijing, China, April 16–20, 2007. Springer, Heidelberg, Germany.

[CG05] Jan Camenisch and Jens Groth. *Group Signatures: Better Efficiency and New Theoretical Aspects*, pages 120–133. Springer, 2005.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *Advances* *in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265, Brighton, UK, April 8–11, 1991. Springer, Heidelberg, Germany.

[DP06] Cécile Delerablée and David Pointcheval. Dynamic fully anonymous short group signatures. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06: 1st International Conference on Cryptology in Vietnam*, volume 4341 of *Lecture Notes in Computer Science*, pages 193–210, Hanoi, Vietnam, September 25–28, 2006. Springer, Heidelberg, Germany.

[ELL$^+$15] Martianus Frederic Ezerman, Hyung Tae Lee, San Ling, Khoa Nguyen, and Huaxiong Wang. A provably secure group signature scheme from code-based assumptions. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 260–285, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GKV10] S. Dov Gordon, Jonathan Katz, and Vinod Vaikuntanathan. A group signature scheme from lattice assumptions. In *ASIACRYPT 2010*, pages 395–412. Springer, 2010.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1069–1083, Austin, TX, 2016. USENIX.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th Annual ACM Symposium on Theory of Computing*, pages 212–219, Philadephia, Pennsylvania, USA, May 22–24, 1996. ACM Press.

[Gro07] Jens Groth. Fully anonymous group signatures without random oracles. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 164–180, Kuching, Malaysia, December 2–6, 2007. Springer, Heidelberg, Germany.

[HBGM18] Andreas Hülsing, Denis Butin, Stefan Gazdag, and Aziz Mohaisen. Xmss: extended hash-based signatures. Technical report, Internet Draft draft-irtf-cfrg-xmss-hash-based-signatures-12. https://datatracker.ietf.org/doc/draft-irtf-cfrg-xmss-hash-based-signatures/, 2018.

[HILL93] Johan Hstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Construction of a pseudo-random generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1993.

[HRB13] Andreas Hülsing, 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 Intelligence Informatics: CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings*, pages 194–208, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Hül13] Andreas Hülsing. W-ots+ – shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *Progress in Cryptology – AFRICACRYPT 2013: 6th International*

Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings, pages 173–188, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 44–61, 1989.

[LLLS13] Fabien Laguillaumie, Adeline Langlois, Benoît Libert, and Damien Stehlé. Lattice-based group signatures with logarithmic signature size. In *ASIACRYPT 2013*, pages 41–61. Springer, 2013.

[LLNW14] Adeline Langlois, San Ling, Khoa Nguyen, and Huaxiong Wang. Lattice-based group signature scheme with verifier-local revocation. In *PKC 2014*, pages 345–361. Springer, 2014.

[LNW15] San Ling, Khoa Nguyen, and Huaxiong Wang. Group signatures from lattices: Simpler, tighter, shorter, ring-based. In Jonathan Katz, editor, *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 427–449. Springer, Heidelberg, Germany, March 30 – April 1, 2015.

[LR86] Michael Luby and Charles Rackoff. *How to Construct Pseudo-random Permutations from Pseudo-random Functions*, pages 447–447. Springer, 1986.

[Mer90] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 218–238, New York, NY, 1990. Springer New York.

[MKF+16] David McGrew, Panos Kampanakis, Scott Fluhrer, Stefan-Lukas Gazdag, Denis Butin, and Johannes Buchmann. State management for hash-based signatures. In *Security Standardisation*, pages 244–260. Springer, 2016.

[MV03] Daniele Micciancio and Salil P. Vadhan. Statistical zero-knowledge proofs with efficient provers: Lattice problems and more. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 282–298, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[NR96] Moni Naor and Omer Reingold. On the construction of pseudo-random permutations: Luby-rackoff revisited. *IACR ePrint*, 1996:11, 1996.

[NSA15] National Security Agency NSA. Commercial national security algorithm suite, 2015. `https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm`.

[NZZ15] Phong Q. Nguyen, Jiang Zhang, and Zhenfeng Zhang. Simpler efficient group signatures from lattices. In *PKC*, pages 401–426. Springer, 2015.

[oSN16] National Institute of Standards and Technology NIST. Post-quantum crypto project, 2016. `http://csrc.nist.gov/groups/ST/post-quantum-crypto/`.

[PQC16] PQCRYPTO. Post-quantum cryptography for long-term security pqcrypto ict-645622, 2016. `https://pqcrypto.eu.org/`.

[Rog04] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption – FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359, New Delhi, India, February 5–7, 2004. Springer, Heidelberg, Germany.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, New Mexico, November 20–22, 1994. IEEE Computer Society Press.