Post-Quantum Group Signatures from Symmetric Primitives

Dan Boneh
Stanford University
dabo@cs.stanford.edu

Saba Eskandarian
Stanford University
saba@cs.stanford.edu

Ben Fisch
Stanford University
bfisch@cs.stanford.edu

Abstract

Group signatures are used extensively for privacy in anonymous credentials schemes and in real-world systems for hardware enclave attestation. As such, there is a strong interest in making these schemes post-quantum secure. In this paper we initiate the study of group signature schemes built only from symmetric primitives, such as hash functions and PRFs, widely regarded as the safest primitives for post-quantum security. We present two constructions in the random oracle model. The first is a group signature scheme satisfying the EPID group signature syntax and security definitions needed for private hardware attestation used in Intel's SGX. The second achieves significantly shorter signatures for many applications, including the use case of remote hardware attestation. While our group signatures for attestation are longer than standard (non-group) post-quantum signatures, they are short enough for applications where the data being signed is large, such as analytics on large private data sets, or streaming media to a trusted display. We evaluate several instantiations of our schemes so that the costs and benefits of these constructions are clear. Along the way we also give improvements to the zero-knowledge Merkle inclusion proofs of Derler et al. (2017).

1 Introduction

Group signatures [23] allow members of a group to anonymously sign messages on behalf of the group, with the added property that a group manager can revoke the credential or possibly strip the anonymity of corrupt members.

In recent years group signatures have become an important privacy mechanism in real-world systems, most prominently in trusted hardware attestation such as Intel's SGX. Group signatures are the essential ingredient in Enhanced Privacy ID, or EPID, used for private attestation [17, 39]. Attestation is a process by which a hardware enclave running on a client device proves the authenticity of its execution environment to a remote party. EPID lets the client device attest, without revealing its identity to the remote party. EPID is based on a group signature scheme [17] that is not post-quantum secure. An adversary who has access to a quantum computer could subvert the attestation process and break a hardware enclave's security in the worst possible way.

In light of the above, there is a strong interest in developing group signatures that are post-quantum secure. The safest way to ensure post-quantum security is to construct a group signature scheme using only symmetric primitives. This is analogous to constructing a standard (non-group) signature scheme from hash functions [9, 18, 21, 45, 46] to obtain a signature scheme whose post-quantum security is virtually assured.

Can we build secure group signatures from symmetric primitives? Bellare et al. [6] give a generic construction from a standard signature scheme, public-key encryption, and a non-interactive zero-knowledge (NIZK) proof. In this generic construction, the group manager adds a member to the group by signing that member's public key. The member can then sign messages anonymously by first using the private key to sign the message, and then computing a NIZK proof of knowledge of both this signature and the group manager's signature on the corresponding public key. This NIZK proof is the member's group signature. With some work, their framework can be adapted to only use symmetric primitives. The NIZK can be built from the "MPC in the Head" technique of Ishai et al. [4,32,38] using random oracles, and the standard signature scheme can also be built from one-way functions and collision-resistant hashing [9, 21, 33, 45]. However, without careful optimization, this generic approach would lead to very inefficient group signatures due to the need for NIZK proofs on complex circuits (the proof size and prover time of these NIZKs is proportional to the number of multiplication gate in the arithmetic circuit representing the statement).

1.1 Our Contributions

We construct a group signature scheme from symmetric primitives, and take a significant step towards reducing the signature size.

Towards this goal, we build two group signature schemes. Our first construction greatly reduces the size of the NIZK statement in the group signature by using PRFs instead of signatures wherever possible. In particular, we are able to replace the inner group member's signature in the generic approach with a PRF evaluation. Our construction does not treat the given primitives as a black-box. Indeed, this is likely necessary by a separation result of Abdalla and Warinschi [1] which rules out black-box constructions for group signatures from one way functions. Consequently, our scheme performs best when instantiated with NIZK-friendly PRFs and CRHFs. In particular, we evaluate the scheme using the LowMC cipher [3].

Next, we show how to significantly improve our group signature by adapting it to the specific real-world use case where signature verification requires an interaction with the group manager to ensure that the signer has not been revoked. We take advantage of this structure to dramatically reduce the signature size by moving many heavy verification steps outside of the NIZK, without compromising anonymity or affecting security. This significantly shrinks the signature size over the first construction.

Along the way, we develop a cute trick for proving membership in a Merkle tree, without revealing the leaf location, using a *third* preimage resistant hash function (Section 5.4). This also provides an improvement to the recent post-quantum accumulators of Derler et al. [26].

Performance and use cases. In Section 5 we discuss options for instantiating our schemes, and measure the sizes of the resulting signatures under different security assumptions. For the circuit sizes needed inside NIZKs in our construction, ZKB++ [21] provides the most efficient proofs. We report sizes for both the Random Oracle and Quantum Random Oracle models [13], and find that our second group signature, designed for attestation, can support groups of over a million members with 3.45 MB signatures at 128-bit post-quantum security. While these signatures are not short, it is important to keep in mind that several megabytes of traffic for attestation is quite acceptable for many applications of trusted hardware, especially where the data transfer needs of the higher-level application dwarf the size of the attestation.

One example is the case of analytics over large private data sets, an area of heavy investment, both in terms of research and financial resources [31, 51]. In this setting, nodes in a distributed network (or the server in a client-server setting) provide a single remote attestation and then exchange a great deal of data. As the quantity of data transferred exceeds millions of database records, the size of the initial attestation ceases to present a major bottleneck.

The case of digital rights management (DRM), for which hardware enclaves such as Intel SGX seem particularly well-suited [24], is another setting where the size of our signatures are acceptable. Consider the common situation where a content provider wishes to stream a movie (easily a few gigabytes in size) to a subscriber while preventing redistribution or unauthorized viewing of copyrighted content [50, 52]. The few additional megabytes of an attestation do not matter next to a film or television series totaling several hundred times their size.

At any rate, our work answers a fundamental question measuring the length and complexity of an optimized group signature from symmetric primitives. Now that we know the answer, developers can decide whether this length is acceptable for their application.

1.2 Additional Related Work

Group Signatures. Group signatures [23] allow members of a group to anonymously produce signatures on behalf of the group, with the added restriction that a group manager has the power to police the behavior of members, e.g. by revoking their group credentials or stripping their anonymity. The most frequently used definitions of group signatures are described by Bellare et al. [6, 7]. Subsequent work on group signatures has led to various schemes, for example, those of Lysyanskaya and Camenisch [19, 20], Boneh et al. [12, 14], and a scheme of Groth [37]. These constructions are not post-quantum secure.

Post-Quantum Signatures and Proofs. Lattice-based cryptography is a popular candidate for post-quantum security. Lattice group signatures were introduced by Gordon et al. [36] and extended in several subsequent works [41–44]. The resulting group signatures are shorter than the ones developed here, but rely on qualitatively stronger post-quantum assumptions.

Another set of post-quantum tools come from the "MPC in the Head" technique [38] for constructing zero-knowledge proofs. This idea has been extended by ZKBoo [32], ZKB++ [21], and Ligero [4]. In particular, Chase et al. use ZKB++ to construt two post-quantum signature schemes Fish and Picnic [21]. The recent development of zk-STARKS [8] opens another avenue to post-quantum zero-knowledge proofs.

Trusted Hardware and Attestation. Hardware enclaves, particularly Intel's SGX [24], have recently been used for a variety of security applications [30, 47]. One of the primary cryptographic components of SGX is its use of direct anonymous attestation, a primitive introduced by Brickell et al. [16] and which relies on group signatures. The EPID attestation mechanism currently in use by SGX, is due to Brickell et al. [17, 39].

2 Preliminaries

Notation. Let $x \leftarrow F(y)$ denote the assignment of the output of $F(y)$ to x, and let $x \overset{\text{R}}{\leftarrow} S$ denote assignment to x of a uniformly random element sampled from set S. We use λ to refer to a security parameter and sometimes omit it if its presence is implicit. The notation $[k]$ represents the set of integers $1, 2, ..., k$, and $\emptyset$ denotes the empty set. We use $\mathcal{A}^H$ to denote that $\mathcal{A}$ has oracle access to some function H. A function $\mathsf{negl}(x)$ is *negligible* if for all $c > 0$, there is an x_0 such that for any $x > x_0$, $\mathsf{negl}(x) < \frac{1}{x^c}$. We omit x if the parameter is implicit. We use $f(x) \approx g(x)$ to mean that for two functions f, g, $|f(x) - g(x)| < \mathsf{negl}(x)$. PPT stands for probabilistic polynomial time. We use the notation $\mathsf{Func}_{\mathcal{A},\mathcal{B}}\langle a, b\rangle$ to refer to a protocol Func between parties $\mathcal{A}$ and $\mathcal{B}$ with inputs a and b, respectively. Finally, we allow algorithms to output $\perp$ to indicate failure.

Standard Primitives. In Appendix A we review the syntax for the standard cryptographic primitives used throughout the paper along with their security properties. In particular, we define pseudorandom functions, secure signatures, commitments, and collision resistant hashing.

Proof Systems. We briefly review the definitions of proof systems that we will need in later sections. The main notion we will use is that of a non-interactive zero knowledge proof of knowledge in the random oracle model. We use the definitions of [28], which modify prior commom reference string-based definitions of non-interactive zero-knowledge for use in the Random Oracle Model.

Definition 1 (Non-interactive Proof System)**.** A non-interactive proof system Π for a relation R consists of prover algorithm that on input x, w outputs a proof π and a verifier algorithm that on input x, π outputs a bit b. We say that (P, V) is correct and sound if it satisfies the following properties:

- $(x, w) \in R \rightarrow V(x, P(x, w)) = 1$
- $(x, w) \notin R \rightarrow \mathsf{Pr}[V(x, P(x, w)) = 1] < \mathsf{negl}$

For convenience and clarity of notation, we use $P(\mathsf{public}(\cdot), \mathsf{private}(\cdot), R)$ to indicate that the public parts of the input to a prover P for relation R correspond to the statement x and that the private parts correspond to the witness w.

The zero-knowledge property [35] informally requires that a proof reveals nothing about (x, w) except that $(x, w) \in R$. Formally, we model this property by describing a *simulator* that can provide a legitimate proof given only x and not w. The simulator S keeps a state st and operates in two modes: one where it generates responses to random oracle queries and another where it generates the actual simulated proof. S takes three inputs: the number 1 or 2 to indicate the mode, the state st, and either a random oracle query q_i or a string x.

Definition 2 (Non-interactive Zero Knowledge [11])**.** Denote with (S_1, S_2) the oracles such that $S_1(q_i)$ returns the first output of $(h_i, \mathsf{st}) \leftarrow S(1, \mathsf{st}, q_i)$ and $S_2(x, w)$ returns the first output of $(\pi, \mathsf{st}) \leftarrow S(2, \mathsf{st}, x)$ if $(x, w) \in R$. We say a protocol $(P^{\mathcal{H}}, V^{\mathcal{H}})$, where $\mathcal{H}$ is a hash function modeled as a random oracle, is a non-interactive zero knowledge (NIZK) proof for R in the random oracle model if there exists a PPT simulator S such that for all PPT distinguishers $\mathcal{D}$ we have

$$\mathsf{Pr}[\mathcal{D}^{\mathcal{H}(\cdot), P^{\mathcal{H}}(\cdot,\cdot)}(1^\lambda) = 1] \approx \mathsf{Pr}[\mathcal{D}^{S_1(\cdot), S_2(\cdot,\cdot)}(1^\lambda) = 1]$$

where both P and S_2 oracles output $\perp$ if $(x, w) \notin R$.

Extractability, informally, is a strengthening of the soundness property that requires any acceptable proof to have an *extractor* algorithm that can efficiently recover w with high probability given the ability to interact with the prover. We refer to Bellare and Goldreich [5] for a full definition.

Definition 3 (Non-interactive Zero Knowledge Proof of Knowledge)**.** We say a non-interactive proof system is a non-interactive zero knowledge proof of knowledge in the random oracle model if it has the correctness, zero-knowledge, and extractability properties defined above.

3 Post-Quantum EPID Group Signatures

In this section we describe and prove the security of our first post-quantum group signature scheme.

3.1 EPID Group Signatures: Definitions

We construct our group signature to the match the syntax and security requirements as defined by Brickel and Li [17]. First, anonymity must ensure that the group manager colluding with any number of group members cannot uncover the identity of the signer. In particular, we do not want the group manager to have a tracing key that lets it compromise a group member's identity from a group signature. Nevertheles, we will later briefly explain how to extend our scheme to achieve traceability, if desired.

Second, we want a revocation property where a group manager can revoke a user's ability to sign by either:

- adding a revoked user's leaked signing key to a revocation list $\mathsf{KEY\text{-}RL}$, or
- adding a revoked user's group signature to a revocation list $\mathsf{SIG\text{-}RL}$.

A user is revoked if its key is included in the list $\mathsf{KEY\text{-}RL}$, or if any of its signatures are included in the list $\mathsf{SIG\text{-}RL}$.

With this setup, we define the syntax and security properties for a group signature scheme as follows.

Definition 4 (Group Signature)**.** A group signature scheme $\mathcal{G}$ involving a group manager $\mathcal{M}$ and n group members, parties $\mathcal{P}_1$ to $\mathcal{P}_n$, consists of algorithms Init, Join, GPSign, $\mathsf{GPVerify}$, $\mathsf{RevokeKey}$ and $\mathsf{RevokeSig}$:

– $(\mathsf{gsk}, \mathsf{gpk}) \leftarrow \mathsf{Init}(1^\lambda)$: This algorithm takes as input a security parameter 1^λ and outputs a key pair $(\mathsf{gsk}, \mathsf{gpk})$.

– $\langle \mathsf{cert}_i, (\mathsf{sk}_i, \mathsf{cert}_i) \rangle \leftarrow \mathsf{Join}_{\mathcal{M},\mathcal{P}_i}\langle (\mathsf{gsk}, \mathsf{gpk}), \mathsf{gpk} \rangle$: This is a protocol between the group manager and a group member $\mathcal{P}_i$ where each party has its keys as input, and both parties get party $\mathcal{P}_i$'s certificate as output. $\mathcal{P}_i$ also gets its secret key sk_i as an output.

– $\perp/\mathsf{sig} \leftarrow \mathsf{GPSign}(\mathsf{gpk}, \mathsf{sk}_i, \mathsf{cert}_i, m, \mathsf{SIG\text{-}RL})$: This algorithm takes as input the public key, a signature revocation list $\mathsf{SIG\text{-}RL}$, and party $\mathcal{P}_i$'s secret key and certificate. The output is a group signature sig.

– $1/0 \leftarrow \mathsf{GPVerify}(\mathsf{gpk}, m, \mathsf{KEY\text{-}RL}, \mathsf{SIG\text{-}RL}, \mathsf{sig})$: This algorithm verifies a group signature sig on a message m given the group public key and key/signature revocation lists $\mathsf{KEY\text{-}RL}$, $\mathsf{SIG\text{-}RL}$. It outputs 1 to accept the signature and 0 to reject it.

– $\mathsf{KEY\text{-}RL} \leftarrow \mathsf{RevokeKey}(\mathsf{gpk}, \mathsf{KEY\text{-}RL}, \mathsf{sk}_i)$: This algorithm adds a secret key sk_i to a key revocation list, so signatures created with this key will no longer be accepted.

– $\mathsf{SIG\text{-}RL} \leftarrow \mathsf{RevokeSig}(\mathsf{gpk}, \mathsf{KEY\text{-}RL}, \mathsf{SIG\text{-}RL}, m, \mathsf{sig})$: This algorithm adds a signature sig to a signature revocation list, so signatures created with the same key as sig will no longer be accepted.

The algorithms must satisfy Correctness (Definition 5), Anonymity (Definition 8), and Unforgeability (Definition 11), which differ only on minor points from those of EPID [17]. We only need one direction of the correctness definition of [17] because unforgeability implies the other direction. That is, we require that if a group member has successfully completed the Join procedure and neither its key nor any of its signatures have been revoked, then that group member's signatures should successfully verify.

Definition 5 (Correctness)**.** Let Σ_i be the set of signatures issued by group member $\mathcal{P}_i$ who has successfully run the Join procedure in group signature scheme $\mathcal{G}$ with security parameter λ. $\mathcal{G}$ is correct if and only if

$$\begin{aligned}&\mathsf{sk}_i \notin \mathsf{KEY\text{-}RL} \wedge \Sigma_i \cap \mathsf{SIG\text{-}RL} = \emptyset \rightarrow \\ &\qquad \Pr[\mathsf{GPVerify}(\mathsf{gpk}, m, \mathsf{KEY\text{-}RL}, \mathsf{SIG\text{-}RL}, \mathsf{GPSign}(\mathsf{gpk}, \mathsf{sk}_i, \mathsf{cert}_i, m, \mathsf{SIG\text{-}RL})) \neq 1] \\ &\qquad\qquad < \mathsf{negl}(\lambda)\end{aligned}$$

Next, we define anonymity via the Anonymity game. Informally, the property of being Anonymous requires that signatures in $\mathcal{G}$ hide the identity of the signer against any coalition of group members (including the group manager) except the signer herself. The definition of anonymity also implies notions of unlinkability between a signer and her signatures.

The strongest possible definition of anonymity, the full-anonymity of Bellare et al [6] which provides anonymity even against the signer herself, cannot be attained in our setting because we wish to support revocation. Revocation is incompatible with full-anonymity because the compromise of a user $\mathcal{P}_i$ in the full-anonymity game would reveal sk_i, the information needed to revoke $\mathcal{P}_i$'s credentials, to the adversary. If revocation were possible, the adversary could then, on its own, build a revocation list that includes only $\mathcal{P}_i$ and use it to determine whether $\mathcal{P}_i$ signed a particular message or not by checking whether the signature verifies.

Definition 6 (Anonymity Experiment)**.** The anonymity experiment denoted by $\mathsf{ANON}[\mathcal{A}, \lambda, b]$ with security parameter λ is played between adversary $\mathcal{A}$ and challenger $\mathcal{C}$ who is given input b.

1. Setup. Adversary $\mathcal{A}$ computes $(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{Init}(1^\lambda)$ and sends gpk to challenger $\mathcal{C}$.
2. Unrestricted Queries. $\mathcal{A}$ is allowed to make as many of the following queries to the $\mathcal{C}$ as it wants:
   - Join. $\mathcal{A}$ requests creation of a new group member $\mathcal{P}_i$. $\mathcal{C}$ runs $\mathsf{Join}_{\mathcal{A},\mathcal{P}_i}\langle(\mathsf{gsk}, \mathsf{gpk}), \mathsf{gpk}\rangle$ with $\mathcal{C}$ playing the role of $\mathcal{P}_i$, so that $\mathcal{A}$ gets cert_i and $\mathcal{C}$ gets $(\mathsf{sk}_i,\ \mathsf{cert}_i)$.
   - Sign. $\mathcal{A}$ requests a signature on a message m from party $\mathcal{P}_i$ relative to a signature revocation list $\mathsf{SIG\text{-}RL}$ of its choosing, constructed from any subset of signatures it has received thus far in the game. $\mathcal{C}$ computes $\mathsf{sig} \leftarrow \mathsf{GPSign}(\mathsf{gpk}, \mathsf{sk}_i, \mathsf{cert}_i, m, \mathsf{SIG\text{-}RL})$ and sends it to $\mathcal{A}$.
   - Corrupt. $\mathcal{A}$ requests the private key of $\mathcal{P}_i$. $\mathcal{C}$ sends sk_i.
3. Challenge. $\mathcal{A}$ sends $\mathcal{C}$ a message m, a signature list $\mathsf{SIG\text{-}RL}$ and two group member numbers i_0 and i_1. $\mathcal{C}$ computes $\mathsf{sig}^* = \mathsf{GPSign}(\mathsf{gpk}, \mathsf{sk}_{i_b}, \mathsf{cert}_i, m, \mathsf{SIG\text{-}RL})$, and sends sig^* to $\mathcal{A}$.
4. Restricted Queries. $\mathcal{A}$ can make additional queries to $\mathcal{C}$ as above.
5. Output. $\mathcal{A}$ outputs b'.

$\mathsf{ANON}[\mathcal{A}, \lambda, b]$ outputs 1 if $b' = b$ ($\mathcal{A}$ wins the game). Otherwise, it outputs 0.

Definition 7 (Admissible Anonymity Adversary)**.** An adversary $\mathcal{A}$ is *admissible* for $\mathsf{ANON}[\mathcal{A}, \lambda, b]$ if it satisfies the following criteria:

– only makes Sign or Corrupt queries on parties that have already participated in a Join and does not make Join queries on parties that have already participated in a Join.

– only sends legitimate certificates cert_i in the join phase (well-formed according to the protocol, e.g. signatures verify, over the correct values).

– chooses parties $\mathcal{P}_{i_0}$ and $\mathcal{P}_{i_1}$ that have already participated in a Join.

– chooses parties $\mathcal{P}_{i_0}$ and $\mathcal{P}_{i_1}$ that have not been corrupted and whose signatures have never appeared in a revocation list.

– makes no Corrupt queries on $\mathcal{P}_{i_0}$ or $\mathcal{P}_{i_1}$ in the "Restricted Queries" stage.

– never includes sig^* in SIG-RL during the "Restricted Queries" stage.

Definition 8 (Anonymous)**.** Group signature scheme $\mathcal{G}$ is Anonymous if no admissible PPT adversary can win the Anonymity game with greater than negligible advantage. That is, if the quantity $|\Pr[\mathsf{ANON}[\mathcal{A}, \lambda, 0] = 1] - \Pr[\mathsf{ANON}[\mathcal{A}, \lambda, 1] = 1]| \leq \mathsf{negl}(\lambda)$ for any admissible PPT $\mathcal{A}$.

Finally, we define unforgeability. Our unforgeability game consists of an adversary who can add arbitrary parties to a group and corrupt arbitrarily many members of a group. Security holds if this adversary cannot forge the signature of an uncorrupted party on a message if its own choosing.

Definition 9 (Unforgeability Experiment)**.** The unforgeability experiment $\mathsf{FORGE}[\mathcal{A}, \lambda]$ with security parameter λ is played between adversary $\mathcal{A}$ and challenger $\mathcal{C}$.

1. Setup. $\mathcal{C}$ computes $(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{Init}(1^\lambda)$ and sends gpk to $\mathcal{A}$. $\mathcal{C}$ creates a set U of corrupted parties and initializes it as $U = \emptyset$.

2. Queries. $\mathcal{A}$ is allowed to make as many of the following queries to $\mathcal{C}$ as it wants.

   – Join. $\mathcal{A}$ requests creation of a new group member $\mathcal{P}_i$. One of two cases follows:

     i. $\mathcal{C}$ runs Join internally, adding a new party $\mathcal{P}_i$ to the group and keeping sk_i, cert_i. $\mathcal{C}$ also sends cert_i to $\mathcal{A}$.

     ii. $\mathcal{C}$ and $\mathcal{A}$ run $\mathsf{Join}_{\mathcal{C},\mathcal{P}_i}\langle(\mathsf{gsk}, \mathsf{gpk}), \mathsf{gpk}\rangle$ with $\mathcal{A}$ playing the role of $\mathcal{P}_i$, so that $\mathcal{C}$ gets cert_i and $\mathcal{A}$ gets (sk_i, cert_i). $\mathcal{A}$ then sends sk_i to $\mathcal{C}$ who then appends i to U.

   – Sign. $\mathcal{A}$ requests a signature on a message m from party $\mathcal{P}_i$ relative to a signature revocation list SIG-RL of its choosing, constructed from any subset of signatures it has received thus far in the game. $\mathcal{C}$ computes $\mathsf{sig} \leftarrow \mathsf{GPSign}(\mathsf{gpk}, \mathsf{sk}_i, \mathsf{cert}_i, m, \mathsf{SIG\text{-}RL})$ and sends it to $\mathcal{A}$.

   – Corrupt. $\mathcal{A}$ requests the secret key of party $\mathcal{P}_i$. $\mathcal{C}$ appends i to U and sends sk_i to $\mathcal{A}$.

3. Forgery. $\mathcal{A}$ outputs a message m^*, revocation lists $\mathsf{KEY\text{-}RL}^*$ and $\mathsf{SIG\text{-}RL}^*$, and a signature sig^*.

$\mathsf{FORGE}[\mathcal{A},\lambda]$ outputs 1 ($\mathcal{A}$ wins the unforgeability game) if $\mathsf{GPVerify}(\mathsf{gpk}, m^*, \mathsf{KEY\text{-}RL}^*, \mathsf{SIG\text{-}RL}^*, \mathsf{sig}^*) = 1$ and for every $i \in U$, either $\mathsf{sk}_i \in \mathsf{KEY\text{-}RL}^*$ or sk_i signs a message in $\mathsf{SIG\text{-}RL}^*$. Otherwise, it outputs 0.

Definition 10 (Admissible Unforgeability Adversary)**.** An adversary $\mathcal{A}$ is *admissible* for $\mathsf{FORGE}[\mathcal{A},\lambda]$ if it satisfies the following criteria:

– only makes Sign or Corrupt queries on parties that have already participated in a Join.

– does not make Join queries on parties that have already participated in a Join.

– only sends legitimate values for sk_i in the join phase. That is, sk_i should correspond to the certificate that $\mathcal{C}$ issues for it.

– does not obtain sig^* by making a Sign query on m^*.

Definition 11 (Unforgeable)**.** Group signature scheme $\mathcal{G}$ is Unforgeable if no admissible PPT adversary can win the Unforgeability game with greater than negligible probability. That is, if the quantity $\Pr[\mathsf{FORGE}[\mathcal{A},\lambda] = 1] \leq \mathsf{negl}(\lambda)$ for any admissible PPT $\mathcal{A}$.

3.2 Group Signature Construction I

Our construction uses a standard (non-group) signature scheme where each group member has its own key pair and a *certificate* from the group manager. Instead of signature keys, however, we construct our scheme so that each group member has a unique PRF secret key that will be used to issue group signatures. As we will see, this leads to significant savings over the general framework of Bellare et al. [6]. We still need a signature scheme for the group manager to produce certificates, but the NIZK proof is done over a circuit that verifies a single signature (the group manager's) along with a few evaluations of the PRF. The signature scheme can be instantiated using a stateful hash-based signature.

Collision Resistant PRF. We state and prove security of our scheme using a function $f : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ that is both a secure PRF and a collision resistant function. In fact, it suffices that f be collision-resistant *on the keyspace*, meaning that for a target input $x \in \mathcal{X}$ chosen by the adversary, it should be hard to find $k_0 \neq k_1 \in \mathcal{K}$ such that $f(k_0, x) = f(k_1, x)$. We explain how to construct an MPC-friendly function with this property in Section 5.

Construction 12 (Group Signature)**.** Our group signature scheme $\mathcal{G}$ = (Init, Join, GPSign, GPVerify, RevokeKey, RevokeSig) with security parameter λ uses a signature scheme $\mathcal{S}$ = (Keygen, Sign, Verify), a proof system $\Pi = (P, V)$, and a PRF f that also serves as a collision-resistant hash function.

– $\mathsf{Init}(1^\lambda)$: Group manager $\mathcal{M}$ runs Keygen(1^λ) to get $(\mathsf{gpk}, \mathsf{gsk})$ and outputs this tuple (gpk is published and gsk kept secret).

– $\mathsf{Join}_{\mathcal{M},\mathcal{P}_i}\langle(\mathsf{gsk}, \mathsf{gpk}), \mathsf{gpk}\rangle$:

  - Group manager $\mathcal{M}$ sends challenge c_i to member $\mathcal{P}_i$.
  - $\mathcal{P}_i$ chooses $\mathsf{sk}_i \xleftarrow{\text{R}} \{0,1\}^\lambda$ and sends $t_i^{\mathsf{join}} = f(\mathsf{sk}_i, c_i)$ back to $\mathcal{M}$.

- $\mathcal{M}$ produces signature $\sigma_i = \mathsf{Sign}(\mathsf{gsk}, (t_i^{\mathsf{join}}, c_i))$, and constructs $\mathsf{cert}_i = (t_i^{\mathsf{join}}, c_i, \sigma_i)$, sending a copy to $\mathcal{P}_i$. If the signature scheme is stateful, then algorithm Join must maintain a counter that is incremented for every user who joins the group.
- The group member's private key is sk_i and both parties get copies of cert_i.

– GPSign(gpk, sk_i, cert_i, m, `SIG-RL`): Compute the following and output sig:

- $r \xleftarrow{R} \{0,1\}^\lambda$
- $t \leftarrow (f(\mathsf{sk}_i, r),\ r)$
- $\pi \leftarrow P\big(\mathsf{public}(\lambda, m, \mathsf{gpk}, t, r, \text{SIG-RL}, \text{KEY-RL}),\ \mathsf{private}(\mathsf{sk}_i, \mathsf{cert}_i),\ R_1\big)$
- $\mathsf{sig} \leftarrow (t, \pi)$.

We define the relation R_1 in the proof of knowledge π for $(\mathsf{sk}_i^*, \mathsf{cert}_i^*)$ to be true when the following statements hold:

- $t = (f(\mathsf{sk}_i^*, r),\ r)$
- $\mathsf{Verify}(\mathsf{gpk}, \sigma_i^*) = 1$
- $t_i^{\mathsf{join}*} = f(\mathsf{sk}_i^*, c_i^*)$
- for each $\mathsf{sig}_j \in \text{SIG-RL}, t_{\mathsf{sig}_j} \neq (f(\mathsf{sk}_i^*, r_{\mathsf{sig}_j}), r_{\mathsf{sig}_j})$

– GPVerify(gpk, m, `KEY-RL`, `SIG-RL`, sig):

- Verify proof π: check $V(\pi) = 1$.
- For each $\mathsf{sk}_j \in$ KEY-RL, check that $t \neq (f(\mathsf{sk}_j, r), r)$.
- Check that $\mathsf{sig} \notin$ SIG-RL.
- Output 1 if all of the above checks return 1; otherwise, output 0.

– RevokeKey(gpk, KEY-RL, sk_i): Return $\text{KEY-RL} \cup \{\mathsf{sk}_i\}$.

– RevokeSig(gpk, KEY-RL, SIG-RL, m, sig): return $\text{SIG-RL} \cup \{\mathsf{sig}\}$ if
GPVerify(gpk, m, KEY-RL, SIG-RL, sig) $= 1$. Otherwise, return SIG-RL.

Performance. We discuss concrete instantiations of this scheme, and their performance in Section 5.

Traceable Signatures. Our approach can also be used to achieve *traceable* signatures. Traceability requires that the group manager have the power to learn the identity of a signer. We presented our scheme without a tracing property in order to guarantee a stronger anonymity property against the group manager, but a similar approach could be used to achieve traceability. The group manager could give each group member a signed secret token sk_i'', and every signature would include the token $t' = (f(\mathsf{sk}_i'', r'), r')$, for a newly picked random r', along with a proof of knowledge of a signature on sk_i''. Now the group manager can trace a signature by trying to reconstruct t' with the value of sk_i'' for each signer, but anonymity will still hold against any other group member. Despite being able to provide both traceability and anonymity, such a construction does not imply public-key encryption via the result of [1] because our anonymity definition does not provide privacy against the signer herself, a requirement of 'full-anonymity' as defined by Bellare et al [6].

3.3 Security proof

We now prove the security of our group signature scheme. Correctness follows almost immediately from the construction with the caveat that we must ensure that the revocation checks do not accidentally cause a signature from a legitimate key to be rejected.

Theorem 13. *Assuming the correctness of signature scheme $\mathcal{S}$ and proof system Π and the pseudorandomness of f, $\mathcal{G}$ is a correct group signature scheme.*

Proof. Correctness follows from the construction. A group member that is not affected by revocations of either form has all the necessary information to produce a signature that will verify, given that the signature and the proof system also have correctness. Then it only remains to show that an unrevoked signature or key sk_i will not "accidentally" satisfy the relation $(f(\mathsf{sk}_i, r),\ r) = (f(\mathsf{sk}_j, r'),\ r')$ for some revoked key sk_j or signature with key/randomness pair (sk_j, r). We call these events $\mathsf{BAD\text{-}KEY}$ and $\mathsf{BAD\text{-}SIG}$ respectively and argue, using the fact that f is a PRF, that they occur with negligible probability over the choices of sk_i, sk_j, r, and r'. We will only show the proof for the case of $\mathsf{BAD\text{-}KEY}$ because the proof for $\mathsf{BAD\text{-}SIG}$ proceeds analogously.

Consider the transcript $T = (t_0, ..., t_k)$ of k (polynomial in λ) responses to queries on f made up of $f(\mathsf{sk}_j, r)$ for all $\mathsf{sk}_j \in \mathsf{KEY\text{-}RL}$ as well as $f(\mathsf{sk}_i, r)$. The event $\mathsf{BAD\text{-}KEY}$ occurs exactly when there exists some j such that $f(\mathsf{sk}_j, r) = f(\mathsf{sk}_i, r)$. We can show that T is indistinguishable from a list of random strings by a series of hybrids, where each successive hybrid replaces the next PRF output with a random string. Each hybrid will be indistinguishable from the next by the security of f. This is the case because an adversary who can distinguish between the list with PRF output t_i and the same list where t_i is replaced by random string t'_i can be given a list with the output of a purported PRF in position i and determine whether that string is the output of a PRF or a truly random function. Call the final hybrid $T' = (t'_0, ..., t'_k)$. The probability of $\mathsf{BAD\text{-}KEY}$ in this hybrid is equal to the probability that the random string $t'_k = t'_j$ for some $j \in [k-1]$. This is negligible in the security parameter λ, completing the proof. □

Next, we show that our group signature scheme provides anonymity. This property follows from the zero-knowledge and pseudorandomness properties of the primitives used in our construction. A full proof follows.

Theorem 14. *Assuming that Π is a zero-knowledge proof system and that f is a PRF, $\mathcal{G}$ is an anonymous group signature scheme.*

Proof. We proceed by a series of hybrids and begin by describing our hybrids. In the following, let x_0 and x_1 be distinct elements of $[N]$, where N is an upper bound on the number of group members. N is necessarily polynomial in λ because the adversary $\mathcal{A}$ is efficient.

- $\mathsf{H}_0[x_0, x_1]$: The real anonymity experiment, $\mathsf{ANON}[\mathcal{A}, \lambda, 0]$ (Definition 6) run with an admissible adversary $\mathcal{A}$ that chooses $i_0 = x_0$ and $i_1 = x_1$ in the Challenge phase of the anonymity game.

- $\mathsf{H}_1[x_0, x_1]$: Same as the previous hybrid, but with the proof of knowledge π always replaced with the output of its simulator. This is indistinguishable from the previous hybrid by the zero-knowledge property of the proof.

– $\mathsf{H}_2[x_0, x_1]$: Same as the previous hybrid, but for group member $\mathcal{P}_{x_0}$, the output of $f(\mathsf{sk}_{x_0}, \cdot)$ is replaced by a random string. This is indistinguishable from the previous hybrid by the PRF security of f.

Indistinguishability between hybrids $\mathsf{H}_0[x_0, x_1]$ and $\mathsf{H}_1[x_0, x_1]$ follows immediately from the zero knowledge property of proof π, so we omit this proof. Next, we prove indistinguishability between the remaining hybrids.

Lemma 1. *Assuming that f is a PRF, the outputs of $\mathsf{H}_1[x_0, x_1]$ and $\mathsf{H}_2[x_0, x_1]$ are indistinguishable.*

Proof. We use an adversary $\mathcal{A}$ that distinguishes between the outputs of $\mathsf{H}_1[x_0, x_1]$ and $\mathsf{H}_2[x_0, x_1]$ to construct an adversary $\mathcal{B}$ that wins the PRF security game. $\mathcal{B}$ acts as the challenger in the anonymity game of $\mathsf{H}_1[x_0, x_1]$ and reproduces it exactly except any queries to $f(sk_{x_0}, \cdot)$ are replaced by queries to the PRF security game challenger. $\mathcal{B}$ then passes on the output of $\mathcal{A}$ as its own output. Note that $\mathcal{B}$ provides a perfect simulation of either $\mathsf{H}_1[x_0, x_1]$ or $\mathsf{H}_2[x_0, x_1]$ depending on whether the PRF challenger uses a PRF or a random function. As such, the output of $\mathcal{B}$ will determine whether it was interacting with a PRF or a random function and win the PRF security game. $\square$

By Lemma 1, we have shown that $\mathsf{H}_0[x_0, x_1]$ is indistinguishable from $\mathsf{H}_2[x_0, x_1]$. We can define corresponding hybrids H'_0-$\mathsf{H}'_2[x_0, x_1]$ starting from $\mathsf{ANON}[\mathcal{A}, \lambda, 1]$. Note that hybrids $\mathsf{H}_2[x_0, x_1]$ and $\mathsf{H}'_2[x_0, x_1]$ describe the same distribution, so we have that hybrids $\mathsf{H}_0[x_0, x_1]$ and $\mathsf{H}'_0[x_0, x_1]$ are indistinguishable. That is, for all admissible PPT adversaries $\mathcal{A}$,

$$|\mathsf{Pr}[\mathsf{ANON}[\mathcal{A}, \lambda, 0] = 1 | i_0 = x_0, i_1 = x_1] - \mathsf{Pr}[\mathsf{ANON}[\mathcal{A}, \lambda, 1] = 1 | i_0 = x_0, i_1 = x_1]| \\ < \mathsf{negl}(\lambda).$$

Now it only remains to show that the probability that $i_0 = x_0, i_1 = x_1$ is polynomial in λ to complete the proof of anonymity. If we choose $x_0, x_1 \xleftarrow{R} [N]$, we will have that $\mathsf{Pr}[x_0 = i_0] = \frac{1}{N}$ and $\mathsf{Pr}[x_1 = i_1] = \frac{1}{N}$ because x_0, x_1 are chosen independently of $\mathcal{A}$. So the probability that $i_0 = x_0, i_1 = x_1$ is $\frac{1}{N^2}$ which is polynomial in λ because N, an upper bound on the group size, is polynomial in λ. Thus we have that

$$\frac{1}{N^2}|\mathsf{Pr}[\mathsf{ANON}[\mathcal{A}, \lambda, 0] = 1] - \mathsf{Pr}[\mathsf{ANON}[\mathcal{A}, \lambda, 1] = 1]| < \mathsf{negl}(\lambda)$$
$$|\mathsf{Pr}[\mathsf{ANON}[\mathcal{A}, \lambda, 0] = 1] - \mathsf{Pr}[\mathsf{ANON}[\mathcal{A}, \lambda, 1] = 1]| < N^2\mathsf{negl}(\lambda).$$

Since $N^2\mathsf{negl}(\lambda)$ is still negligible in λ, this completes the proof. $\square$

Finally, we show that our group signature scheme is unforgeable. Intuitively, unforgeability comes from the fact that in order to produce a valid signature without the endorsement of the group manager, an attacker must guess a group member's unrevoked secret key.

Theorem 15. *Assuming that Π is a zero knowledge proof of knowledge proof system, $\mathcal{S}$ is an unforgeable signature scheme, that f is a PRF, and that f is additionally a collision-resistant hash function, $\mathcal{G}$ is an unforgeable group signature scheme.*

Proof. We proceed by a series of hybrids and begin by describing our hybrids:

– H_0: The real unforgeability game, $\mathsf{FORGE}[\mathcal{A}, \lambda]$ run with an admissible adversary $\mathcal{A}$.

– H_1: Same as previous hybrid, but we also run the extractor on each hidden value in the proof of knowledge π from the forgery and output 0 (i.e. adversary loses) if the extractor fails. This is indistinguishable from the previous world by the extractability property of the proof of knowledge.

– H_2: Same as the previous hybrid, but we output 0 (i.e. adversary loses) if the signature of the cert_i extracted from the forgery is not from a certificate issued by the group manager. This is indistinguishable from the previous world by the unforgeability of signature scheme $\mathcal{S}$.

– H_3: Same as previous hybrid, but we abort if there exists a $j \in U$, that is, a party $\mathcal{P}_j$ in the set of corrupted group members, such that for cert_i extracted from the forgery, $t_i^{\mathsf{join}} = f(\mathsf{sk}_j, c_i)$. This is indistinguishable from the previous world by the collision-resistance of f.

Indistinguishability between hybrids H_0 and H_1 follows immediately from the extractability of π. We therefore omit this proof. Next, we prove indistinguishability between the remaining hybrids.

Lemma 2. *Assuming that $\mathcal{S}$ is an unforgeable signature scheme, the outputs of H_1 and H_2 are indistinguishable.*

Proof. Note that so long as the signature of the cert_i extracted from adversary $\mathcal{A}$'s forgery corresponds to a certificate issued by the group manager, the outputs of H_1 and H_2 are identical, so the only case in which the two distributions differ is when $\mathsf{Verify}(\mathsf{gpk}, t_i^{\mathsf{join}}) = 1$ and cert_i was not issued by the group manager (and the adversary wins). Call this event F. We will show that F occurs with at most negligible probability.

We build an adversary $\mathcal{B}$ for $\mathcal{S}$'s unforgeability game that wins with non-negligible probability if $\mathsf{Pr}[F] > \mathsf{negl}(\lambda)$. $\mathcal{B}$ acts as the challenger in the group signature unforgeability game H_2 and reproduces it exactly except any signing queries to $\mathsf{Sign}(gsk, \cdot)$ are sent to the unforgeability game for $\mathcal{S}$. As its forgery, $\mathcal{B}$ outputs the value t_i^{join} extracted from adversary $\mathcal{A}$'s group signature forgery.

$\mathcal{B}$ wins $\mathcal{S}$'s unforgeability game exactly when event F occurs. Thus, if F occurs with more than negligible probability, $\mathcal{B}$ breaks the unforgeability of signature scheme $\mathcal{S}$. Since $\mathcal{S}$ is an unforgeable signature scheme, F must occur with at most negligible probability, so the outputs of H_1 and H_2 must only differ with at most negligible probability.

□

Lemma 3. *Assuming that f is a collision-resistant hash function, the outputs of H_2 and H_3 are indistinguishable.*

Proof. Note that so long as there is no $j \in U$ such that for cert_i extracted from the forgery, $t_i^{\mathsf{join}} = f(\mathsf{sk}_j, c_i)$, the outputs of H_2 and H_3 are identical, so the only case in which the two distributions differ is when there does exist such a j and the adversary $\mathcal{A}$ successfully outputs a forgery. Call this event F. We will show that F occurs with at most negligible probability.

Since it is possible for $j \in U$ to be revoked by key or by signature, we will show only the case where the group member $\mathcal{P}_j$ is revoked by key. The case for revocation by signature is analogous. Let t_i be the value of t extracted from the $\mathcal{A}$'s forgery. In order for event F to occur, the adversary must produce a value sk_i such that $(f(\mathsf{sk}_i, r),\ r) = t_i \neq (f(\mathsf{sk}_j, r),\ r)$ and $f(\mathsf{sk}_i, c_j) = t_j^{\mathsf{join}} = f(\mathsf{sk}_j, c_j)$.

We build an adversary $\mathcal{B}$ that breaks the collision-resistance of f when event F occurs with greater than negligible probability. $\mathcal{B}$ acts as the challenger in the group signature unforgeability game H_2. At the conclusion of the game, if there exists a group member $\mathcal{P}_j$ as defined in H_3, $\mathcal{B}$ outputs the values (sk_i, c_j) and (sk_j, c_j), where sk_i is extracted from $\mathcal{A}$'s forgery and sk_j by a linear search over compromised group member keys, as its candidate collision for f. Otherwise, it fails to output a collision.

$\mathcal{B}$ outputs a successful collision on f whenever event F occurs. Thus, if F occurs with greater than negligible probability, $\mathcal{B}$ breaks the collision-resistance of f on t_j^{join}. Since f is collision-resistant, F must occur with at most negligible probability, so the outputs of H_2 and H_3 must only differ with negligible probability.

□

By Lemmas 2 and 3, we have shown that $\mathsf{FORGE}[\mathcal{A}, \lambda]$ is indistinguishable from H_3. Now we will show how to use an adversary $\mathcal{A}$ who successfully outputs a forgery in H_3 with non-negligible probability to construct an adversary $\mathcal{B}$ that breaks the PRF security of f, completing the proof.

Adversary $\mathcal{B}$ begins by picking a value $n^* \xleftarrow{\mathrm{R}} [N]$, where N is an upper bound on the number of members in the group. Then $\mathcal{B}$ acts as the challenger in the anonymity game of H_3 and reproduces it exactly except any queries to $f(sk_{n^*}, \cdot)$ are replaced by queries to the PRF security game challenger. Let $\mathsf{F}_{\mathsf{PRF}}$ be the function computed by the PRF adversary. If the cert_i extracted from the $\mathcal{A}$'s forgery is not equal to cert_{n^*}, $\mathcal{B}$ aborts. Otherwise, for the value r used in $\mathcal{A}$'s forgery, $\mathcal{B}$ queries the PRF adversary on r to get response $\mathsf{F}_{\mathsf{PRF}}(r)$. If $\mathsf{F}_{\mathsf{PRF}}(r) = f(\mathsf{sk}_{n^*}, r)$ $\mathcal{B}$ outputs 1 (interacting with PRF). Otherwise, it outputs 0.

Now we argue that $\mathcal{B}$ successfully distinguishes between a PRF and a random function. First, suppose for the certificate cert_i extracted from the forgery, that $\mathsf{cert}_i = \mathsf{cert}_{n^*}$, so $\mathsf{F}_{\mathsf{PRF}}(c_{n^*}) = f(\mathsf{sk}_{\mathsf{n}^*}, c_{n^*})$. If $\mathcal{B}$ is interacting with a random function, $\mathcal{B}$ will output 0 with high probability because a random function only collides with $f(\mathsf{sk}_{n^*}, \cdot)$ on r with negligible probability. On the other hand, if $\mathcal{B}$ is interacting with a PRF with key $\mathsf{sk}_{n'}$, there are two cases:

– $\mathsf{sk}_{n'} = \mathsf{sk}_{n^*}$: In this case, $\mathcal{B}$ always outputs 1.

– $\mathsf{sk}_{n'} \neq \mathsf{sk}_{n^*}$: Then $\mathsf{f}(\mathsf{sk}_{n'}, c_{n^*}) = f(\mathsf{sk}_{\mathsf{n}^*}, c_{n^*})$ is a collision that violates the collision-resistance of f. Since all the algorithms involved in finding this collision are efficient, this case must only occur with negligible probability.

All that remains is to show that $\mathsf{cert}_i = \mathsf{cert}_{n^*}$ with non-negligible probability. Since cert_i corresponds to a certificate issued by the group manager with all but negligible probability (in which case $\mathcal{A}$ would abort early), we are assured that $i \in [N]$. Since n^* is chosen independently of $\mathcal{A}$'s choice of i, there is a $\frac{1}{N}$ chance that $n^* = i$, which is certainly non-negligible in λ because there can only be polynomially many group members created by $\mathcal{A}$.

□

4 Practical Post-Quantum Group Signatures for Attestation

Attestation schemes (such as that used in Intel SGX [24,39]) involve an attestation service "in the loop" every time an attestation needs to be verified despite the fact that this is not necessary for the underlying group signature scheme. Put in terms of the group signature setting, every time a group signature is verified, there is a step that involves contacting the group manager to get an updated revocation list. This requirement means that frequent contact between the group manager and group members should be possible. In this section, we leverage the continuing availability of the group manager in the attestation setting to build significantly smaller post-quantum group signatures.

The number of gates required to verify the signature on a group member's certificate by far outweighs that of other components in the proof of knowledge included in each signature of the scheme from Section 3. Moving this verification outside of the proof would dramatically shrink signature sizes, and this is exactly what we do. In our modified scheme, each group member's certificate is a leaf in a Merkle tree. The group manager signs the root of the tree and provides each group member a membership proof as part of the Join process. Now the group manager's signature can be verified outside the proof of knowledge because the group manager's signature on the publicly known root of the tree leaks nothing about the identity of a particular signer. Instead of verifying a signature inside a proof, the signer now only needs to verify a Merkle inclusion proof – an operation that requires a much smaller circuit, compared to verifying a hash-based stateful signature.

The above modification, while greatly improving efficiency, introduces a critical security flaw in the model where each group member registers with the group manager once and then begins creating signatures: a new Merkle tree root will need to be published by the group manager each time a group member joins! As an immediate consequence, group members joining earlier suffer from smaller anonymity sets. Even worse, a curious group manager could issue a sequence of Merkle roots where each tree only included a valid credential for one group member, uniquely identifying the member's signatures.

Fortunately, the continuing contact between group members and the group manager enforced by attestation in practice enable effective mitigations for the concerns listed above. Group members can periodically "re-join" the group to update the Merkle root relative to which they provide membership proofs, thereby increasing the size of their anonymity sets. In practice, we can ensure that subsequent Merkle roots issued by the group manager only ever add new credentials to the group and never omit previous ones by using a *Merkle consistency proof* such as the one proposed by the Certificate Transparency standard [40] and proven secure by Dowling et al [27]. We model the Merkle trees used in our proofs as accumulators with zero-knowledge membership proofs and discuss how we instantiate this primitive with an improved construction in Section 5.

4.1 Definitions

In this section we define accumulators and group signatures for attestation. We begin with a special case of the formalization of accumulators by [25].

Definition 16 (Accumulator)**.** A static accumulator is a tuple of efficient algorithms (AGen, AEval, $\mathsf{AWitCreate}$, $\mathsf{AVerify}$, $\mathsf{AProveCon}$, $\mathsf{ACheckCon}$) which are defined as follows:

- $\mathsf{AGen}(1^\lambda)$: This algorithm takes a security parameter λ and returns a public key $\mathsf{pk}_\wedge$.
- $\mathsf{AEval}(\mathsf{pk}_\wedge, \mathcal{X})$: This deterministic algorithm takes a key $\mathsf{pk}_\wedge$ and a set $\mathcal{X}$ to be accumulated and returns an accumulator $\Lambda_\mathcal{X}$.
- $\mathsf{AWitCreate}(\mathsf{pk}_\wedge, \Lambda_\mathcal{X}, \mathcal{X}, x_i)$: This algorithm takes a key $\mathsf{pk}_\wedge$, an accumulator $\Lambda_\mathcal{X}$, the set $\mathcal{X}$, and a value x_i. It returns $\perp$ if $x_i \notin \mathcal{X}$ and a witness wit_{x_i} for x_i otherwise.
- $\mathsf{AVerify}(\mathsf{pk}_\wedge,\ \Lambda_\mathcal{X},\ \mathsf{wit}_{x_i},\ x_i)$: This algorithm takes a public key $\mathsf{pk}_\wedge$, an accumulator $\Lambda_\mathcal{X}$, a witness wit_{x_i}, and a value x_i. It returns 1 if wit_{x_i} is a witness for $x_i \in \mathcal{X}$ and 0 otherwise.

We require accumulators to be correct, meaning that $\mathsf{AVerify}$ will accept an honestly generated witness for $x_i \in \mathcal{X}$. We also require a soundness property dubbed *collision-freeness*, formally defined below.

Definition 17 (Collision Freeness)**.** An accumulator is collision free if for all PPT adversaries $\mathcal{A}$, we have that

$$\Pr[\mathsf{pk}_\wedge \leftarrow \mathsf{AGen}(1^\lambda), (\mathsf{wit}^*_{x_i}, x^*_i, \mathcal{X}^*) \leftarrow \mathcal{A}(\mathsf{pk}_\wedge) | \\ \mathsf{AVerify}(\mathsf{pk}_\wedge, \Lambda^*, \mathsf{wit}^*_{x_i}, x^*_i) = 1 \wedge x^*_i \notin \mathcal{X}^*] \leq \mathsf{negl}(\lambda)$$

where $\Lambda^* \leftarrow \mathsf{Eval}_{r^*}(\mathsf{pk}_\wedge, \mathcal{X}^*)$.

The setting of group signatures for attestation largely leaves the security definitions of Section 3 unaffected up to changes in syntax, so we present the updated syntax for clarity of presentation and omit statements of the security properties. The only notable change is that in both security games, the adversary can now choose to have a group member run the new $\mathsf{GARejoin}$ at any time it chooses.

Definition 18 (Group Signature for Attestation)**.** A group signature scheme $\mathcal{GA}$ for attestation involving a group manager $\mathcal{M}$ and n group members parties $\mathcal{P}_1$ to $\mathcal{P}_n$ consists of algorithms GAInit, GAJoin, $\mathsf{GARejoin}$, GASign, $\mathsf{GAVerify}$, $\mathsf{RevokeKey}$ and $\mathsf{RevokeSig}$. In the following, $\mathcal{X}$ represents a set, Λ represents a static accumulator representing $\mathcal{X}$, and σ_Λ is a signature on Λ.

– $(\mathsf{gsk}, \mathsf{gpk}) \leftarrow \mathsf{GAInit}(1^\lambda)$: This algorithm takes as input a security parameter 1^λ and outputs a key pair $(\mathsf{gsk}, \mathsf{gpk})$.

– $\langle(\mathsf{cert}_i, \Lambda, \sigma_\wedge), (\mathsf{sk}_i, \mathsf{cert}_i, \Lambda, \sigma_\wedge)\rangle \leftarrow \mathsf{GAJoin}_{\mathcal{M},\mathcal{P}_i}\langle(\mathsf{gsk}, \mathsf{gpk}, \mathcal{X}), \mathsf{gpk}\rangle$: This is a protocol between the group manager and a group member $\mathcal{P}_i$ where each party has its keys as input, and the group manager also has the set $\mathcal{X}$ of group member credentials. Both parties get party $\mathcal{P}_i$'s certificate, an accumulator value Λ, and a signature $\sigma_\wedge$ on Λ from the group manager as output. $\mathcal{P}_i$ also gets its secret key sk_i as an output.

– $\langle(\mathsf{cert}_i, \Lambda, \sigma_\wedge), (\mathsf{cert}_i, \Lambda, \sigma_\wedge)\rangle \leftarrow \mathsf{GARejoin}_{\mathcal{M},\mathcal{P}_i}\langle(\mathsf{gsk}, \mathsf{gpk}, \mathcal{X}, \Lambda, \sigma_\wedge), (\mathsf{gpk}, \mathsf{cert}_i)\rangle$: This is a protocol between the group manager and a group member $\mathcal{P}_i$ where the group manager has the group key pair, a set of user credentials $\mathcal{X}$, an accumulator Λ for $\mathcal{X}$, and a signature $\sigma_\wedge$ on Λ as inputs, and group member $\mathcal{P}_i$ has the group public key and its certificate as inputs. Both parties get an updated certificate for $\mathcal{P}_i$ as well as the accumulator value Λ and signature $\sigma_\wedge$ as outputs.

– $\perp/\mathsf{sig} \leftarrow \mathsf{GASign}(\mathsf{gpk}, \mathsf{sk}_i, \mathsf{cert}_i, m, \mathsf{SIG\text{-}RL}, \Lambda, \sigma_\wedge)$: This algorithm takes as input the public key, party $\mathcal{P}_i$'s secret key and certificate, a signature revocation list $\mathsf{SIG\text{-}RL}$, an accumulator Λ, and a signature $\sigma_\wedge$ on Λ from the group manager. The output is a group signature sig.

– $1/0 \leftarrow \mathsf{GAVerify}(\mathsf{gpk}, m, \mathsf{KEY\text{-}RL}, \mathsf{SIG\text{-}RL}, \mathsf{sig})$: This algorithm verifies a group signature sig on a message m given the group public key and key/signature revocation lists $\mathsf{KEY\text{-}RL}$, $\mathsf{SIG\text{-}RL}$. It outputs 1 to accept the signature and 0 to reject it.

– $\mathsf{KEY\text{-}RL} \leftarrow \mathsf{GARevokeKey}(\mathsf{gpk}, \mathsf{KEY\text{-}RL}, \mathsf{sk}_i)$: This algorithm adds a secret key sk_i to a key revocation list, so signatures created with this key will no longer be accepted.

- $\mathsf{SIG\text{-}RL} \leftarrow \mathsf{GARevokeSig}(\mathsf{gpk}, \mathsf{KEY\text{-}RL}, \mathsf{SIG\text{-}RL}, m, \mathsf{sig})$: This algorithm adds a signature sig to a signature revocation list, so signatures created with the same key as sig will no longer be accepted.

In order to capture the security guarantees of our new setting, namely the fact that anonymity only applies relative to the anonymity set of users with the same merkle root, we add the following admissibility criterion for anonymity adversaries.

Definition 19 (Admissible Anonymity Adversary for Attestation)**.** An adversary $\mathcal{A}$ is *admissible* for $\mathsf{ANON}[\mathcal{A}, \lambda, b]$ if it satisfies the following criteria:

- It is an admissible anonymity adversary as in Definition 7.
- It chooses parties $\mathcal{P}_{i_0}$ and $\mathcal{P}_{i_1}$ that produce signatures relative to the same accumulator Λ.

4.2 Group Signature Construction II

The full construction of the modified group signature scheme appears below. Structurally similar to the construction in Section 3, the main changes involve the introduction of a post-quantum accumulator and the resulting restructuring of what needs to be proven inside/outside the proof of knowledge π.

Construction 20 (Group Signature for Attestation)**.** Our group signature scheme for attestation $\mathcal{GA}$ = (GAInit, GAJoin, $\mathsf{GARejoin}$, GASign, $\mathsf{GAVerify}$, $\mathsf{GARevokeKey}$, $\mathsf{GARevokeSig}$) with security parameter λ uses a signature scheme $\mathcal{S}$ = (Keygen,Sign,Verify), a proof system $\Pi = (P, V)$, a PRF f that also serves as a collision-resistant hash function, and an accumulator $\mathcal{A}c$ = (AGen, AEval, $\mathsf{AWitCreate}$, $\mathsf{AVerify}$).

- $\mathsf{GAInit}(1^\lambda)$: Group manager $\mathcal{M}$ runs Keygen(1^λ) to get $(\mathsf{pk_{gp}}, \mathsf{sk_{gp}})$ and runs $\mathsf{AGEN}(1^\lambda)$, to get $\mathsf{pk}_\wedge$. It outputs public key $\mathsf{gpk} = (\mathsf{pk_{gp}}, \mathsf{pk}_\wedge)$ and secret key $\mathsf{gsk} = \mathsf{sk_{gp}}$.
- $\mathsf{GAJoin}_{\mathcal{M},\mathcal{P}_i}\langle(\mathsf{gsk}, \mathsf{gpk}, \mathcal{X}), \mathsf{gpk}\rangle$:
  - Group manager $\mathcal{M}$ sends challenge c_i to member $\mathcal{P}_i$.
  - $\mathcal{P}_i$ picks $\mathsf{sk}_i \xleftarrow{R} \{0,1\}^\lambda$ and sends $t_i^{\mathsf{join}} = f(\mathsf{sk}_i, c_i)$ back to $\mathcal{M}$.
  - $\mathcal{M}$ defines $x_i = (t_i^{\mathsf{join}}, c_i)$, sets $\mathcal{X} = \mathcal{X} \cup x_i$, sets $\Lambda = \mathsf{AEval}(\mathsf{pk}_\wedge, \mathcal{X})$, and produces signature $\sigma_\wedge = \mathsf{Sign}(\mathsf{gsk}, \Lambda)$. Next, $\mathcal{M}$ creates $\mathsf{wit}_{x_i} = \mathsf{AWitCreate}(\mathsf{pk}_\wedge, \Lambda, \mathcal{X}, x_i)$ and constructs $\mathsf{cert}_i = (x_i, \mathsf{wit}_{x_i})$, sending a copy to $\mathcal{P}_i$ along with Λ and $\sigma_\wedge$.
  - The group member's private key is sk_i and both parties get copies of cert_i, Λ, and $\sigma_\wedge$.
- $\mathsf{GARejoin}_{\mathcal{M},\mathcal{P}_i}\langle(\mathsf{gsk}, \mathsf{gpk}, \mathcal{X}, \Lambda, \sigma_\wedge), (\mathsf{gpk}, \mathsf{cert}_i)\rangle$:
  - $\mathcal{P}_i$ sends cert_i to $\mathcal{M}$.
  - First, $\mathcal{M}$ verifies the signature in cert_i and aborts if verification fails. Then it creates a new $\mathsf{wit}_{x_i} = \mathsf{AWitCreate}(\mathsf{pk}_\wedge, \Lambda, \mathcal{X}, x_i)$ and constructs the updated $\mathsf{cert}_i = (x_i, \mathsf{wit}_{x_i})$, sending a copy to $\mathcal{P}_i$ along with Λ and $\sigma_\wedge$.
  - $\mathcal{P}_i$ updates its values of cert_i, Λ, and $\sigma_\wedge$.

– $\mathsf{GASign}(\mathsf{gpk}, \mathsf{sk}_i, \mathsf{cert}_i, m, \mathsf{SIG\text{-}RL}, \Lambda, \sigma_\wedge)$: Compute the following and output sig:

  - $\mathsf{Verify}(\mathsf{pk}_{\mathsf{gp}}, \sigma_\wedge, \Lambda)$ (abort if it outputs 0)
  - $r \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$
  - $t = (f(\mathsf{sk}_i, r), r)$
  - $\pi = P(\mathsf{public}(\lambda, m, \mathsf{gpk}, t, r, \mathsf{SIG\text{-}RL}, \ \mathsf{KEY\text{-}RL}, \Lambda), \ \mathsf{private}(\mathsf{sk}_i, \mathsf{cert}_i), \ R_2)$
  - $\mathsf{sig} = (t, \pi, \Lambda, \sigma_\wedge)$.

  We define R_2 as a relation in the proof of knowledge of $(\mathsf{sk}_i^*, \mathsf{cert}_i^*)$ such that the following statements hold:

  - $t = (f(\mathsf{sk}_i^*, r), r)$
  - $\mathsf{AVerify'}(\mathsf{pk}_\wedge, \Lambda, \mathsf{wit}_{x_i}^*, (t_i^{\mathsf{join}*}, c_i^*))$
  - $t_i^{\mathsf{join}*} = f(\mathsf{sk}_i^*, c_i^*)$
  - for each $\mathsf{sig}_j \in \mathsf{SIG\text{-}RL}, t_{\mathsf{sig}_j} \neq (f(\mathsf{sk}_i^*, r_{\mathsf{sig}_j}), r_{\mathsf{sig}_j})$

– GAVerify(gpk, m, `KEY-RL`, `SIG-RL`, sig):

  - Verify signature $\sigma_\wedge$: check $\mathsf{Verify}(\mathsf{pk}_{\mathsf{gp}}, \sigma_\wedge, \Lambda) = 1$
  - Verify proof π: check $V(\pi) = 1$.
  - For each $\mathsf{sk}_j \in \mathsf{KEY\text{-}RL}$, check that $t \neq (f(\mathsf{sk}_j, r), r)$.
  - Check that $\mathsf{sig} \notin \mathsf{SIG\text{-}RL}$.
  - If all of the above checks return 1, output 1. Else, output 0.

– GARevokeKey(gpk, KEY-RL, sk_i): Return $\mathsf{KEY\text{-}RL} \cup \mathsf{sk}_i$.

– GARevokeSig(gpk, KEY-RL, SIG-RL, m, sig): If GAVerify(gpk, m, KEY-RL, SIG-RL, sig) $= 1$, return $\mathsf{SIG\text{-}RL} \cup \mathsf{sig}$. Otherwise, return SIG-RL.

Performance. We discuss concrete instantiations of this scheme, and their performance in Section 5.

4.3 Security Proofs

Correctness and anonymity proofs for $\mathcal{GA}$ are almost completely unchanged from our standard group signature scheme, so we only state the corresponding theorems and do not repeat the proofs.

Theorem 21. *Assuming the correctness of signature scheme $\mathcal{S}$, proof system Π, and accumulator $\mathcal{A}c$, as well as the pseudorandomness of f, $\mathcal{GA}$ is a correct group signature scheme.*

Theorem 22. *Assuming that Π is a zero-knowledge proof system and that f is a PRF, $\mathcal{GA}$ is an anonymous group signature scheme.*

The unforgeability proof strongly resembles the proof of the original scheme, but we no longer need to extract the group manager's signatures before verifying them. Instead, we extract and verify a membership proof. Since the two proofs are otherwise identical, we omit a full proof of security and only state the hybrids used in the proof, sketching the transitions between them.

Theorem 23. *Assuming that Π is a proof system for zero-knowledge proofs of knowledge, $\mathcal{S}$ is an unforgeable signature scheme, that f is a PRF, that f is additionally a collision-resistant hash function, and that $\mathcal{A}c$ is a collision-free (sound) accumulator, $\mathcal{GA}$ is an unforgeable group signature scheme.*

Proof (sketch). We proceed by a series of hybrids:

- H_0: The real unforgeability game, $\mathsf{FORGE}[\mathcal{A}, \lambda]$ run with an admissible adversary $\mathcal{A}$.
- H_1: Same as the previous hybrid, but we output 0 (i.e. adversary loses) if the signature $\sigma_\wedge$ received by the verifier in any ReJoin is not on an accumulator value Λ output by $\mathcal{M}$ in a Join protocol. This is indistinguishable from the previous world by the unforgeability of signature scheme $\mathcal{S}$.
- H_2: Same as the previous hybrid, but we output 0 (i.e. adversary loses) if the signature $\sigma_\wedge$ in the forgery is not for the value of Λ included in the forgery or if Λ is not an accumulator value output by $\mathcal{M}$ in a Join protocol. This is indistinguishable from the previous world by the unforgeability of signature scheme $\mathcal{S}$.
- H_3: Same as previous hybrid, but we also run the extractor on each hidden value in the proof of knowledge π from the forgery and output 0 (i.e. adversary loses) if the extractor fails. This is indistinguishable from the previous world by the extractability property of the proof of knowledge.
- H_4: Same as the previous hybrid, but we output 0 (i.e. adversary loses) if the membership proof wit_{x_i} extracted from the forgery does not correspond to a value of $x_i \in \mathcal{X}$, the set accumulated by Λ. This is indistinguishable from the previous world by the collision-freeness of the accumulator $\mathcal{A}c$, as the proof wit_{x_i} could be used to break collision-freeness if it were produced with greater than negligible probability.
- H_5: Same as previous hybrid, but we abort if there exists a $j \in U$, that is, a party $\mathcal{P}_j$ in the set of corrupted group members, such that for cert_i extracted from the forgery, $t_i^{\mathsf{join}} = f(\mathsf{sk}_j, c_i)$. This is indistinguishable from the previous world by the collision-resistance of f.

Indistinguishability proofs between hybrids are analogous to those of group signature scheme $\mathcal{G}$, as is the final argument for unforgeability from Hybrid H_5. $\square$

5 Instantiation of Protocols

We have now described and proven the security of our constructions, but the post-quantum security of each construction relies on the existence of post-quantum secure instantiations of the various primitives required. In particular we require a PRF that is also a collision-resistant hash function, a signature scheme, zero knowledge proofs of knowledge (ZKPoKs), and an accumulator. In this section we describe options for instantiating each primitive under different security assumptions about the underlying ciphers used and report the signature sizes of our instantiated schemes in both the Random Oracle (RO) and Quantum Random Oracle (QRO) models [13].

5.1 Zero Knowledge Proofs of Knowledge

In principle, standard symmetric primitives (AES, SHA) suffice for post-quantum security so long as we double our security parameters. However, our schemes uses these primitives in a non-black box manner by running them inside of a ZKPoK. In particular, the following ZKPoKs contribute significantly to signature sizes:

1. ZKPoK of a PRF key k such that $f(k, r) = t$, for a PRF that is collision-resistant on its key space.

2. ZKPoK of a signature σ on a message m such that $\mathsf{Verify}(m, \sigma) = 1$ for a post-quantum signature scheme $\mathcal{S} = (\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Verify})$.

3. ZKPoK of membership of element x_i in accumulator Λ for set $\mathcal{X}$.

We restrict our choice of ZKPoK proof system to those systems which rely only on symmetric primitives. This includes works following the "MPC in the Head" approach of Ishai et al [38] – ZKBoo [32], ZKB++ [21], and Ligero [4] – as well as zk-STARKs [8]. Although Ligero and zk-STARKs offer proofs asymptotically sublinear in the size of the circuit to be proven, a preliminary analysis suggested that, for our relatively small proof circuits, ZKB++ provides the smallest signature sizes in practice without requiring heavy computing costs for the signer. Moreover, ZKB++ has proofs of security in both the Random Oracle and Quantum Random Oracle models, whereas Ligero and zk-STARKs only have proofs in the classical RO model. As such, we choose to instantiate our signatures and measure signature size using ZKB++ as our underlying ZKPoK system.

In ZKB++ [21], the underlying statement to be proven is represented as an arithmetic circuit over $\mathsf{GF}(2)$, and the proof size is proportional to the multiplicative complexity (i.e., number of AND gates) in the circuit. The most important practical consideration in signature schemes is signature size; therefore our main criterion in instantiating the PRF and outer signature scheme is to minimize their multiplicative complexity over $\mathsf{GF}(2)$.

5.2 PRF and Collision-Resistant Hash Function

Recently the ciphers LowMC [3] and MiMC [2] have been proposed as alternatives to AES that have significantly lower multiplicative complexity as arithmetic circuits over finite fields.[1] Although relatively new and less extensively studied, these ciphers were shown to resist statistical cryptanalytic attacks, similar to other state-of-the-art designs. A number of works have already proposed using LowMC as the best candidate to-date for instantiating ciphers inside ZKB++-style proofs [21, 26]. The most recent public version of the LowMC cipher with parameters set for 128-bit post-quantum security (256-bit key, 256-bit block size) involves only 1374 AND gates, a significant improvement over the 7616 AND gates in AES-256 [3].

Derler et. al [26] also suggest using the LowMC round function in the sponge framework (as described in [2]) to construct a collision-resistant hash function with low multiplicative complexity. However, since only a collision-resistant compression function on a fixed message length is needed (rather than full-blown indifferentiability from a random oracle), we propose applying the much simpler Davies-Meyer transformation to the LowMC cipher. Collision resistance of Davies-Meyer is

[1]LowMC optimizes multiplicative complexity over GF(2) while MiMC optimizes complexity over larger finite fields. In ZKB++ the underlying circuit is represented in GF(2), which is why we prefer LowMC.

proved in the ideal cipher model [15], which is only marginally stronger than the security assumption underlying the sponge transformation. Given an ideal cipher $E(k, x)$ on equal sized key and message space, the Davies-Meyer compression function is $H(m_1||m_2) = E(m_1, m_2) \oplus m_2$. For a collision-resistant PRF we would use $F(k, x) = E(k, x) \oplus x$; as long as E is a PRF then F is also a PRF. Note that the multiplicative complexity of F is the same as E. To obtain a PRF that is collision-resistant only on its keyspace we need only rely on the slightly weaker assumption that E is indistinguishable from a random permutation on a fixed key k_{fix}: define $\Pi(y) = E(k_{\mathsf{fix}}, y)$, and define $F'(k, x) = \Pi(E(k, x)) \oplus E(k, x)$. (The inner evaluation of $E(k, x)$ ensures the PRF property while $\Pi(y) \oplus y$ is collision resistant as a special case of Davies-Meyer).

5.3 Post-Quantum Signature Scheme

Choices for post-quantum signatures that do not rely on stronger lattice assumptions include Merkle signatures [45], Goldreich's stateless signatures [33], SPHINCS signatures [9], or the Fish signatures of Chase et al [21]. The recent literature on post-quantum signatures has focused on optimizing signature size. When using signatures outside of proofs (in our construction of group signatures for attestation) we propose using SPHINCS, which has the smallest signature size. However, since our main group signature construction involves verifying the group manager's post-quantum signature inside a ZKPoK, there we care about optimizing the arithmetic multiplicative complexity of signature verification rather than the signature size.

We examine two options for instantiating the group manager's signature scheme for signatures used inside a ZKPoK: one using stateful Merkle signatures, and other using Goldreich's stateless signatures.

Stateful Merkle signatures The signer runs a signature setup that generates a large number of one-time signature (OTS) keypairs. We would use Lamport signatures from one-way functions (instantiated with LowMC) for the OTS. The Lamport signature private key consists of 256 pairs of pseudorandom 256-bit strings the public key consists of the 256 pairs of outputs generated by applying the one-way function to each private key string. The signer finalizes the setup by computing a Merkle tree (using a 2-to-1 collision resistant compression function) over the OTS public keys at the leaves of the tree and publishing the root as the public verification key. Signing a message involves singing the message with one of the leaf OTS keys and proving membership of this OTS key in the Merkle tree. The signer needs to maintain state to ensure that no OTS key is used more than once. The stateful requirement is not prohibitive in the setting of managing a group of trusted hardware platforms. The preprocessing of a tree of up to 2^{30} members would take under a day on modern commodity hardware and would require the server to use only several GB of storage.

Stateless Goldreich signatures Instead of maintaining state in the Merkle signature scheme above, the signer could choose an OTS key at random. This requires squaring the size of the tree to make collisions unlikely. For a group of 2^{30} members storing a tree of size 2^{60} keys would be prohibitively expensive. However, Godlreich's scheme provides a way to generate this tree pseudorandomly from a small seed. In this scheme, the signer pseudorandomly generates an OTS keypair for each node of the tree, which can be done by evaluating a PRF on the index of the tree node. The OTS public key at the root of the tree is the overall public key. The OTS key pair on each node of the tree is used to sign the hash of the public keys on each of its two child nodes. To sign a message a random leaf is selected and the signature includes the OTS signatures along the path from this leaf to the root, where each signature signs either a child public key or the actual message at the leaf.

5.4 Reducing Circuit Size for Membership Proofs

As mentioned in Section 4, we will use Merkle trees to instantiate our accumulators. A recent work of Derler et al [26] points out, however, that the circuit used to verify standard Merkle inclusion proofs differs based on the path from the Merkle root to the leaf x_i. The dependence arises based on whether the hash at depth j of the tree becomes the left or right input the hash at depth $j-1$. This dependence of the $\mathsf{AVerify}$ circuit on i must be removed in order to generically create a zero-knowledge inclusion proof with some zero-knowledge proof system. They suggest a modification to the standard inclusion proof that allows the same circuit to verify inclusion regardless of the index i whose inclusion is proven. The idea is as follows: suppose x_i resides in a subtree rooted at internal node a and that a has sibling and parent nodes b and c, respectively. At each level of the Merkle tree, instead of simply calculating $h(a, b)$ and only comparing the result to the root, they evaluate the expression $c = h(a, b) \vee c = h(a, b)$ and reject the inclusion proof if it is not satisfied. This allows the construction of a circuit $\mathsf{AVerify}$' with a fixed ordering of inputs to each hash function, since as long as one ordering of inputs matches the node at the next level of the tree, correctness will hold. The cost of this transformation is an extra hash evaluation, an equality check, and a logical OR for each level of the tree.

We propose a solution that eliminates the need for equality checks at each level of the tree and replaces the OR with an XOR, allowing smaller and more efficient zero-knowledge membership proofs. Our idea is to replace the hash function h already used in computing the merkle root with a modified function $h'(x, y) = h(x, y) \oplus h(y, x)$. Using h' in place of h proves that the input x_i is a d^{th} preimage of the merkle root for a tree of depth d without any dependence on the position i of x_i among the tree's leaves. Of course, h' is trivially neither collision-resistant nor second preimage resistant, as a swapping of the inputs x and y results in the same output. Below we prove that h' provides a *third* preimage resistance property and can be used to build the inclusion proofs we desire.

First, we recall the standard notion of a second preimage resistant hash function.

Definition 24 (Second Preimage Resistance [15])**.** We say a hash function H defined over $(\mathcal{M}, \mathcal{T})$ is second preimage resistant if given a random $m \in \mathcal{M}$, it is difficult to find a different $m' \in \mathcal{M}$ such that $H(m) = H(m')$.

Now we define a *third* preimage resistant hash function and show that h' satisfies this property under certain conditions.

Definition 25 (Third Preimage Resistance)**.** We say a hash function H defined over $(\mathcal{M}, \mathcal{T})$ is third preimage resistant if given a random $m \in \mathcal{M}$ and a different $m' \in \mathcal{M}$ such that $H(m) = H(m')$, it is difficult to find an $m'' \in \mathcal{M}$ such that $H(m'') = H(m) = H(m')$.

Lemma 4. *Assuming the hash function $h : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ is a random function, the hash function $h'(x, y) = h(x, y) \oplus h(y, x)$ for $x, y \in \mathcal{M}$ is third preimage resistant, provided $x \neq y$.*

Proof. $h'(x, y)$ admits a trivial collision $h'(y, x)$. We argue it is hard to find any other collision unless $x = y$ (since $h'(x, x) = 0$ for all x). To find a third preimage of $h'(x, y)$ an adversary must produce w, z such that either $h'(w, z) = h'(x, y)$ and either $w \neq x$ or $z \neq y$. Since h is a random function and $(x, y), (y, x), (w, z), (z, w)$ are all distinct tuples, $h(x, y)$, $h(y, x)$, $h(w, z)$, and $h(z, w)$ will all be independently random strings. The probability that $h(x, y) \oplus h(y, x) = h(w, z) \oplus h(z, w)$ is therefore negligible in the length $|x| + |y|$. Therefore no efficient adversary can find a third preimage for h'. □

In order to replace h with h' in our merkle tree construction and retain security for the circuit $\mathsf{AVerify}$', we only need to show that we will have no leaves $x||y$ in the accumulator such that $x = y$. Fortunately, since the elements in the accumulator for our particular case are challenge/response pairs $(f(sk_i, c_i), c_i)$ that serve as group member credentials (where f is collision-resistant and a PRF), the probability that $x = y$ is negligible in our setting.

Practically, using our new circuit $\mathsf{AVerify}$' reduces the number of equality checks needed inside a ZKPoK from $2\log_2(N)$ (where N is the group size) to 1. Additionally, $\log_2(N)$ OR gates are replaced with XORs which do not increase proof size.

5.5 Signature Sizes

As discussed above, we instantiate our signatures using LowMC, Merkle signatures (inside the ZKPoK), SPHINCS signatures (outside the ZKPoK), ZKB++, and Merkle tree accumulators with our modified membership proof circuit.

Figure 1 shows the sizes for our modified group signatures for various group sizes under (1) the assumption that LowMC is and ideal cipher and (2) the assumption that LowMC with a public fixed key is a random permutation. Figure 2 presents the same information, but uses the Unruh transform [48] instead of the Fiat-Shamir transform [29] to make the ZKB++ proof noninteractive. The Fiat-Shamir transform is proven secure in the Random Oracle model but only sometimes retains security in the Quantom Random Oracle model [13,49]. As visible from the figures, groups of size up to 2^{20} could use post-quantum signatures of size 6.74MB (3.45MB in RO model) under our scheme, a sufficiently small size for attestation in applications with heavy data transfer requirements. The same signatures instantiated with AES-256 would require 33.8MB (16.9MB in RO model).

For comparison, our signature sizes are smaller than the recent ring signatures of Derler et al [26], which require at least 10.4MB (5.26MB in RO Model) for signatures in a ring of 2^{20} members[2], despite providing a more elaborate functionality. The improvement comes from our new accumulator membership proofs, as the accumulator constitutes the most costly component of both constructions.

Our general-purpose group signatures require 216.82MB for signatures in a group of size 2^{30} assuming LowMC is an ideal cipher (110.81MB in QRO Model), a much larger value than the variation designed for attestation. This motivates the question of how to generalize the specialized version of our construction to apply to a wider range of use-cases. The most limiting component of the modified construction is the requirement that the group manager remain available to parties after joining in order for them to run $\mathsf{GAReJoin}$ and update their membership proofs. If this procedure could be made non-interactive, many more use cases could be satisfied by the modified group signatures. This could be achieved if there were a way to update an accumulator membership proof given only the old and new accumulator values along with the old membership proof. We leave

[2]This size represents an optimized version of the ring signatures instantiated assuming LowMC is an ideal cipher. The original Derler et al paper claims slightly larger signatures of size 11.88MB (8MB in RO Model) for this ring size.

Signature Sizes in RO Model

Group Size	Ideal Cipher	Random Permutation
2^7	1.37MB	2.28MB
2^{10}	1.85MB	3.21MB
2^{20}	3.45MB	6.31MB
2^{30}	5.05MB	9.41MB
2^{40}	6.65MB	12.5MB

Figure 1: Signature sizes for construction II under various security assumptions on LowMC, using the Fiat-Shamir transform [29] to make proofs of knowledge noninteractive.

Signature Sizes in QRO Model

Group Size	Ideal Cipher	Random Permutation
2^7	2.64MB	4.45MB
2^{10}	3.59MB	6.30MB
2^{20}	6.74MB	12.5MB
2^{30}	9.89MB	18.6MB
2^{40}	13.0MB	24.8MB

Figure 2: Signature sizes for construction II under various security assumptions on LowMC, using the Unruh transform [48] to make proofs of knowledge noninteractive.

the task of constructing an accumulator with such an update procedure from symmetric primitives as an open problem.

6 Conclusion

We presented a general-purpose post-quantum group signature scheme as well as a construction of a specialized group signature variant designed for trusted hardware enclave attestation. We also gave an analysis of the concrete sizes of our signatures based on the best possible instantiations with current tools and showed that our group signatures for attestation can achieve sizes acceptable for use in some applications.

Group signatures play an important role in modern trusted hardware architectures. Making these signatures post-quantum secure is an important goal, and doing it from symmetric primitives alone is the safest way to do it. We hope that this work will spur further research on this question that will further reduce the signature size.

Acknowledgments

We would like to thank David Wu for several helpful conversations. This work is supported by NSF, the DARPA/ARL SAFEWARE project, the Simons foundation, and a grant from ONR. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

References

[1] Michel Abdalla and Bogdan Warinschi. On the minimal assumptions of group signature schemes. In *ICICS*, pages 1–13, 2004.

[2] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT*, pages 191–219, 2016.

[3] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT*, pages 430–454, 2015.

[4] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104, 2017.

[5] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO*, pages 390–420, 1992.

[6] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT*, pages 614–629, 2003.

[7] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, pages 136–153, 2005.

[8] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018.

[9] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In *EUROCRYPT*, pages 368–397, 2015.

[10] Manuel Blum. Coin flipping by telephone. In *Advances in Cryptology: A Report on CRYPTO 81, CRYPTO 81, IEEE Workshop on Communications Security*, pages 11–15, 1981.

[11] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

[12] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.

[13] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *ASIACRYPT*, pages 41–69, 2011.

[14] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pages 168–177. ACM, 2004.

[15] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography.* 2017.

[16] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *CCS*, pages 132–145, 2004.

[17] Ernie Brickell and Jiangtao Li. Enhanced privacy ID from bilinear pairing. *IACR Cryptology ePrint Archive*, 2009:95, 2009.

[18] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.

[19] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *SCN*, pages 268–289, 2002.

[20] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, pages 56–72, 2004.

[21] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *CCS*, pages 1825–1842, 2017.

[22] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982.

[23] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.

[24] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[25] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA*, pages 127–144, 2015.

[26] David Derler, Sebastian Ramacher, and Daniel Slamanig. Post-quantum zero-knowledge proofs for accumulators with applications to ring signatures from symmetric-key primitives. *IACR Cryptology ePrint Archive*, 2017.

[27] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and certificate transparency. In *ESORICS*, pages 140–158, 2016.

[28] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In *INDOCRYPT*, pages 60–79, 2012.

[29] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[30] Ben Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, and Sergey Gorbunov. IRON: functional encryption using intel SGX. In *CCS*, pages 765–782, 2017.

[31] Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. Sok: Cryptographically protected database search. In *IEEE Symposium on Security and Privacy (Oakland)*, pages 172–191, 2017.

[32] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security*, pages 1069–1083, 2016.

[33] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[34] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *CRYPTO*, pages 276–288, 1984.

[35] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

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

[37] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[38] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[39] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel software guard extensions: Epid provisioning and attestation services.

[40] B. Laurie, A. Langley, and E. Kasper. Certificate transparency. RFC 6962, June 2013.

[41] Benoît Libert, San Ling, Fabrice Mouhartem, Khoa Nguyen, and Huaxiong Wang. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In *ASIACRYPT*, pages 373–403, 2016.

[42] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In *EUROCRYPT*, pages 1–31, 2016.

[43] San Ling, Khoa Nguyen, and Huaxiong Wang. Group signatures from lattices: Simpler, tighter, shorter, ring-based. In *PKC*, pages 427–449, 2015.

[44] San Ling, Khoa Nguyen, Huaxiong Wang, and Yanhong Xu. Lattice-based group signatures: Achieving full dynamicity with ease. In *ACNS*, pages 293–312, 2017.

[45] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 369–378, 1987.

[46] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 33–43. ACM, 1989.

[47] Kartik Nayak, Christopher W. Fletcher, Ling Ren, Nishanth Chandran, Satya Lokam, Elaine Shi, and Vipul Goyal. Hop: Hardware makes obfuscation practical. In *NDSS*.

[48] Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In *EUROCRYPT*, pages 755–784, 2015.

[49] Dominique Unruh. Post-quantum security of fiat-shamir. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 65–95, 2017.

[50] Miao Yu, Virgil D. Gligor, and Zongwei Zhou. Trusted display on untrusted commodity platforms. In *CCS*, pages 989–1003, 2015.

[51] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI*, pages 283–298, 2017.

[52] Zongwei Zhou, Virgil D. Gligor, James Newsome, and Jonathan M. McCune. Building verifiable trusted path on commodity x86 computers. In *IEEE Symposium on Security and Privacy (Oakland)*, pages 616–630, 2012.

A Standard Primitives: Definitions

Definition 26 (Pseudorandom Function [34]). Let $F : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be an efficiently computable, length-preserving keyed function. We say that F is a pseudorandom function (PRF) if for all probabilistic polynomial time distinguishers D,

$$|\mathsf{Pr}[D^{F_k}(1^n) = 1] - \mathsf{Pr}[D^{f_n}(1^n) = 1]|$$

is negligible where $k \leftarrow \{0,1\}^n$ is chosen uniformly at random and f_n is chosen uniformly at random from the set of functions mapping n-bit strings to n-bit strings.

Definition 27 (Collision-Resistant Hash Function [15]). We say that a hash function h over $(\mathcal{M}, \mathcal{T})$ is collision resistant if for all efficient uniform adversaries $\mathcal{A}$, $\mathcal{A}$ cannot output two messages m_0, m_1 such that $h(m_0) = h(m_1)$ with greater than negligible probability.

Definition 28 (Signature). A signature scheme $\mathcal{S}$ is a triple $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ of PPT algorithms which are defined as follows:

– $\mathsf{KeyGen}(1^\lambda)$: This algorithm takes a security parameter λ as input and outputs a secret (signing) key sk and a public (verification) key pk.

– $\mathsf{Sign}(\mathsf{sk}, m)$: This algorithm takes a secret key sk and a message m as input and outputs a signature σ.

– $\mathsf{Verify}(\mathsf{pk},\ m,\ \sigma)$: This algorithm takes a public key pk, a message m, and a signature σ as input and outputs a bit $b \in \{0,1\}$.

The two properties we require from signatures are correctness and unforgeability. Correctness requires that $\mathsf{Verify}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1$. Unforgeability (informally) requires that a computationally bounded adversary cannot forge a signature it has not seen with greater than negligible probability. Formally, we define unforgeability using the notion of existential unforgeability under a chosen message attack.

Definition 29 (Existential Unforgeability Under Chosen Message Attack (EUF-CMA))**.** A signature scheme $\mathcal{S}$ is existentially unforgeable under chosen message attacks if for all PPT adversaries $\mathcal{A}$, we have that:

$$\begin{gathered}\mathsf{Pr}[(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda), (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})| \\ \mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1 \wedge m^* \notin \mathcal{Q}^{\mathsf{Sign}}] \leq \mathsf{negl}(\lambda)\end{gathered}$$

where $\mathcal{Q}^{\mathsf{sign}}$ keeps track of the queries made by the adversary to the signing oracle.

Definition 30 (Commitment Scheme [10])**.** A commitment scheme consists of two algorithms $(\mathsf{Com}, \mathsf{Vrfy})$ with the following properties:

– $\mathsf{Com}(m,\ r)$: On input of a message m and randomness r, the commitment algorithm outputs a commitment c.

– $\mathsf{Vrfy}(c,\ m,\ r)$: On input a commitment c, message m, and randomness r, Vrfy outputs a bit $b \in \{0,1\}$.

We require that computationally secure commitments satisfy the following three properties:

– Correctness: $\mathsf{Vrfy}(\mathsf{Com}(m,\ r),\ m,\ r)$=1.

– Hiding: For every message pair m, m', no computationally bounded adversary can distinguish between commitments to m and m' with greater than negligible advantage.

– Binding: No computationally bounded adversary can produce a commitment c such that $\mathsf{Vrfy}(c, m, r) = \mathsf{Vrfy}(c, m', r') = 1$ for $m \neq m'$ with greater than negligible probability.