A New Hash-based Enhanced Privacy ID Signature Scheme

Liqun Chen* Changyu Dong† Nada EL Kassem*
Christopher J. P. Newton* Yalan Wang*

Abstract

The elliptic curve-based Enhanced Privacy ID (EPID) signature scheme is broadly used for hardware enclave attestation by many platforms that implement Intel Software Guard Extensions (SGX) and other devices. This scheme has also been included in the Trusted Platform Module (TPM) specifications and ISO/IEC standards. However, it is insecure against quantum attackers. While research into quantum-resistant EPID has resulted in several lattice-based schemes, Boneh *et al.* have initiated the study of EPID signature schemes built only from symmetric primitives. We observe that for this line of research, there is still room for improvement. In this paper, we propose a new hash-based EPID scheme, which includes a novel and efficient signature revocation scheme. In addition, our scheme can handle a large group size (up to 2^{60} group members), which meets the requirements of rapidly developing hardware enclave attestation applications. The security of our scheme is proved under the Universal Composability (UC) model. Finally, we have implemented our EPID scheme, which, to our best knowledge, is the first implementation of EPID from symmetric primitives.

1 Introduction

The concept of Enhanced Privacy ID (EPID). Like group signatures [19] and Direct Anonymous Attestation (DAA) signatures [11], EPID is a type of anonymous signature scheme, which allows users in a group to sign messages such that the signatures can be verified using a group public key, and the actual signers' identities are not revealed to the verifier (beyond the fact that they belong to the group). The difference between these three types of signatures is the methods that they handle traceability and revocation. A group signature is traceable, meaning that given a signature, an authorised entity, namely a group tracer, can find the actual signer. However, to avoid the tracer being a privacy bottleneck, both DAA and EPID do not support traceability.

*University of Surrey, {liqun.chen, nada.elkassem, c.newton, yalan.wang}@surrey.ac.uk
†Guangzhou University, changyu.dong@gzhu.edu.cn

Revocation is defined as "the withdrawal of the power of a signing key that has been granted". A typical revocation method is using a revocation list. Different types of revocation lists are considered in anonymous signatures. When a group public key is on the revocation list, the entire group can be revoked. Group signatures, DAA, and EPID all support this type of revocation. When a certain group member's key material is on the revocation list, this member can be revoked. Some group signature schemes put a part of the membership credential on the revocation list, e.g. [9]. Both DAA and EPID can use a private key revocation list, such that if a signer's private key is revealed, this signer is revoked. This is not a powerful revocation since a signer's private key is known only by the signer and there is no way to force a malicious signer to revoke itself. To avoid this weakness, both DAA and EPID further have signature-based revocation. In DAA, this is through linkability, i.e., two DAA signatures signed by the same signer are linked if they use the same basename. For example, to access a digital service, signers are required to use a given basename from the service provider, who based on the user's behaviour can build a local signature revocation list. In EPID, a signature can directly be put on the revocation list, and an EPID signature includes an extra Non-Interactive Zero-Knowledge Proof (NIZKP) to demonstrate that none of the signatures in the list was signed using the current signing key, while it does not leak any extra information about a signer who has not been revoked. This is a unique and powerful feature of EPID. The challenge is to make the cost of the NIZKP as low as possible.

Related work. The idea and first scheme of EPID were proposed by Brickell and Li [12, 15]. They aim to build a new DAA scheme with enhanced revocation capabilities. Their security notion of EPID is a modification of the security model of the original DAA scheme [11] by replacing linkability with signature-based revocation. The security of their scheme is based on the strong RSA and the decisional Diffie-Hellman assumptions. Shortly after that, Brickell and Li [13, 14] proposed a more efficient EPID scheme using bilinear pairings, and its security is based on the strong Diffie-Hellman and the decisional Diffie-Hellman assumptions. The computational and communicational costs of both proof generation and verification are linear to the size of the signature revocation list. The authors expected that the revocation list would be rather small. Later, Dall *et al.* [25] identified that the Intel SGX EPID implementation leaks sensitive key information via a cache side channel. Based on our understanding, this side-channel attack does not show any design flaw in EPID schemes. Recently, Faonio *et al.* [31] introduced a subversion resilient EPID (SR-EPID) scheme, which provides the same functionality and security guarantees of the original EPID, despite potentially subverted hardware. Their scheme uses also bilinear pairings. The security of their scheme is based on the External Diffie-Hellman assumption, i.e., both pairing input groups hold the decisional Diffie-Hellman assumption. As for security analysis, Muhammad *et al.* worked on the formal method-based security analysis of EPID-based remote attestation [46]. They made use of a fully automated formal approach using a popular automatic symbolic protocol verifier, ProVerif [7], to specify and verify the EPID-based attestation process in the Intel SGX. El Kassem *et al.* [30] presented a new security model for EPID

in the Universal Composability (UC) framework.

The pairing-based EPID scheme [14] is used for hardware enclave attestation by many platforms that equip Intel Software Guard Extensions (SGX). This scheme is included in ISO/IEC standards [37]. A TPM implementation of EPID is included in the TPM 2.0 library specifications [47]. All these standard EPID schemes make use of elliptic curves and their security is based on Diffie-Hellman-related problems, so they are not quantum-safe. Designing a post-quantum EPID scheme has drawn the cryptographic community's attention. Solutions based on lattices or hash functions have appeared in the literature.

In 2019, El Kassem *et al.* [30] proposed the first EPID scheme based on lattices. The security of their scheme relies on the ring Short Integer Solution (SIS) and Learning With Error (LWE) hard problems. Their scheme is proven secure under the UC framework. In 2023, Biswas *et al.* [6] proposed another lattice-based EPID scheme. The security of their scheme relies on the hardness of the standard SIS problem. They adopted an updatable Merkle tree accumulator to ensure that any group member can join or be revoked dynamically at any time. Again in 2023, Chen *et al.* [20] proposed a new lattice-based Secure Device Onboard EPID (SDO-LEPID) scheme, their construction is based on the lattice-based DAA protocol in [23]. They implemented both the lattice-based EPID scheme [29] and their proposed scheme and claimed that their new EPID scheme is more efficient. However, we observe that the performance of their protocol can be further improved by relying on more efficient and compact lattice-based zero-knowledge proofs such as in [10, 42].

Boneh *et al.* [8] initiated the study of EPID from symmetric primitives and proposed two EPID schemes. Their schemes rely on hash functions, pseudorandom functions (PRFs), and Non-Interactive Zero-Knowledge Proofs (NIZKPs) only. There is no known quantum attack on these primitives, so their schemes are quantum-resistant. More specifically, the first scheme makes use of an NIZKP of PRFs and the NIZKP is instantiated by MPC-in-the-Head (MPCitH); considering a real-world use case of remote hardware attestation, the second scheme uses a Merkle-tree-based accumulator to arrange group credentials, that allows to reduce the signature size by moving many heavy verification steps outside of the NIZKP. It is claimed that the maximum group size can reach 2^{40} theoretically. They have not implemented their schemes although discussed key sizes and signature sizes for different sizes of a group.

Our contributions. The focus of this paper is to improve the existing work on EPID from symmetric primitives. We observe that the following aspects have not yet been completed: (1) find an efficient method to prove an unrevealed signature key has not been used to create any signatures in a signature revocation list; (2) let a symmetric-based EPID scheme handle a large group size, aiming to 2^{60}; (3) prove the security of a symmetric-based EPID scheme under the Universal Composability (UC) model; and (4) implement a symmetric-based EPID scheme and check their performance precisely. The contribution of this paper is to complete these aspects. We propose a new hash-based EPID scheme, which includes a novel and efficient signature revocation scheme. Our scheme can handle a large group size (up to 2^{60} group members), which meets the requirements

of rapidly developing hardware enclave attestation applications. The security of our scheme is proved under the UC model. Finally, we have implemented our EPID scheme, which, to our best knowledge, is the first implementation of EPID from symmetric primitives.

Outline of the paper. The remaining part of this paper is arranged as follows: § 2 describes relevant preliminaries, § 3 presents the proposed EPID construction, § 4 and 5 provide security notions and proofs, § 6 discusses our implementation result and compares it with the existing post-quantum EPID implementation results, and § 7 concludes this paper. For the sake of completion, we provide the security analysis of a group membership credential scheme in Appendix A.

2 Preliminaries

2.1 Hash-based signatures

In a hash-based signature scheme, a private key is composed of a series of randomly generated strings, while the corresponding public key is obtained by applying hash functions to the private key. Early hash-based signature schemes, such as the Lamport scheme [41] and the Winternitz scheme [43], were one-time signatures (OTS), meaning that each key pair can only be used to sign a single message. The Merkle signature scheme [43] is the first hash-based few-time signatures (FTS). It generates several OTS key pairs and aggregates their public keys using a Merkle tree. The root of the tree serves as the overall public key. Every signature uses one OTS private key, and it is comprised of the corresponding OTS and the Merkle tree authentication path for the OTS public key. The verifier can verify multiple signatures using only the Merkle tree root.

These signature schemes are characterized as stateful, as the signer is required to maintain a state containing information such as the number of signed messages and the keys utilized. In comparison, more recent FTS schemes, such as FORS [4], is stateless as they utilize a large set of secret random strings that can be obtained from a pseudorandom function applied to the private key. Signatures are then generated by selecting elements from the set based on the message to be signed. While each signature discloses some secret strings in the set, the set size is large, and the number of signatures can be controlled to make it infeasible to forge a signature by mixing and matching secret strings from previously generated signatures. Building on the top of FORS, SPHINCS+ [4] is a more powerful stateless hash-based signature scheme. It employs a hypertree, i.e., a tree of trees, to organize OTS and FTS key pairs. Each SPHINCS+ signature constitutes a chain of signatures, with the initial signature Σ_0 being generated from the message, and each subsequent signature Σ_i being a signature of the public key that verifies the preceding signature Σ_{i-1}. By using the root public key, the authenticity of the signature chain can be verified. Although SPHINCS+ also has an upper limit on the number of signatures that can be generated per key pair, it can be set to an extremely large value (up to 2^{64}),

making it highly unlikely to reach this limit in practical scenarios. SPHINCS+ has been chosen as one of the three digital signature schemes by the National Institute of Standards and Technology (NIST) to become a part of its post-quantum cryptographic standard [45].

2.2 MPC-in-the-Head based signatures

Ishai et. al. [35] introduced the idea of Zero-Knowledge Proofs (ZKP) based on "Multi-Party Communication in the Head" (MPCitH). Given a public value x, the prover proves knowing a witness w such that $f(w) = x$. To do so, the prover simulates, by itself, an MPC protocol between m parties that realizes f, in which w is secretly shared as an input to the parties, and commits to the views and internal state of each party. Next, the verifier challenges the prover to open a subset of these commitments, checks them, and decides whether to accept or not. If the MPC realizes f properly, this protocol is complete, meaning a valid statement will always be accepted. The protocol is also zero-knowledgeable because only the views and internal states of a subset of the parties are available to the verifier, and by the privacy guarantee of the underlying MPC protocol, no information about w can be leaked. For soundness, if the prover tries to prove a false statement, then the joint views of some of the parties must be inconsistent, and with some probability, the verifier can detect that. The soundness error of a single MPC run can be high, but by repeating this process independently enough times, the soundness error can be made negligible. The interactive zero-knowledge proofs can be made Non-Interactive Zero-Knowledge Proofs (NIZKP) through techniques such as the Fiat-Shamir transformation.

There are many solutions for constructing MPCitH NIZKPs, e.g., IKOS [35], ZKBoo [34], ZKB++ [18], KKW [39], Ligero++ [5], Limbo [27], BBQ [26], Banquet [2], BN++ [38], Rainer [28], AIM [40], Aurora [3], VOLE-in-the-Head [1] and Polaris [33]. They follow the same paradigm but are different in the underlying MPC protocols and have different concrete/asymptotic efficiency. In this paper, to describe our scheme, we do not need to touch the low-level details, hence we will use MPCitH (for Boolean circuits) in an abstract way. We will use the following syntax to describe a NIZKP:

$$\pi = \mathcal{P}\{(\texttt{public params}); (\texttt{witness}) | \texttt{relation to be proved}\}$$

For example, to prove the same key sk is used in two different instantiations of a pseudorandom function F with different data inputs, we write:

$$\pi = \mathcal{P}\{(C_1, P_1), (C_2, P_2)); (sk) | C_1 = F(sk, P_1) \wedge C_2 = F(sk, P_2)\}$$

MPCitH has been used to generate signature schemes from a symmetric key setting. The first scheme is Picnic [48, 18, 49], in which the secret signing key is k and the public verification key is a pair (c, p), and the key pair satisfy the equation $c = E(k, p)$ where E is a block cipher, k is a secret key, and p and c are respectively a plaintext and ciphertext block. To enhance security and improve performance, more MPCitH-based signature schemes have been developed, e.g., Banquet [2], Rainer [28], FAEST [1], AIM [40] and Peron [24].

Our EPID scheme makes use of this type of signature scheme, and any scheme that holds the Existential Unforgeability under a Chosen Message Attack (EUF-CMA) can be used. More specifically, we create a public and private key pair from a keyed pseudo-random function written as $y = F(sk, x)$, where sk is a secret signing key and the corresponding public verification key is $pk = (x, y)$. Signing a message m essentially is to generate a non-interactive MPCitH proof of knowing sk:

$$\pi = \mathcal{P}\{(x, y); (sk)|y = F(sk, x)\}(m)$$

2.3 M-FORS and F-SPHINCS+ signatures

To construct an EPID scheme from symmetric primitives, we need to select symmetric setting-based group membership credentials. A credential is a signature on a group member's key generated by the issuer. Following the research on hash-based group signatures [22] and hash-based DAA [21], we choose an F-SPHINCS+ signature as a group credential. The F-SPHINCS+ signature scheme is a modification of the SPHINCS+ signature scheme [4]. As a result, F-SPHINCS+ signatures are more MPCitH friendly than SPHINCS+ signatures. As depicted in Fig. 1, F-SPHINCS+ signatures make use of a hyper-tree that is a tree of M-FORS trees. We now recall the description of M-FORS and F-SPHINCS+ from [21, 22].

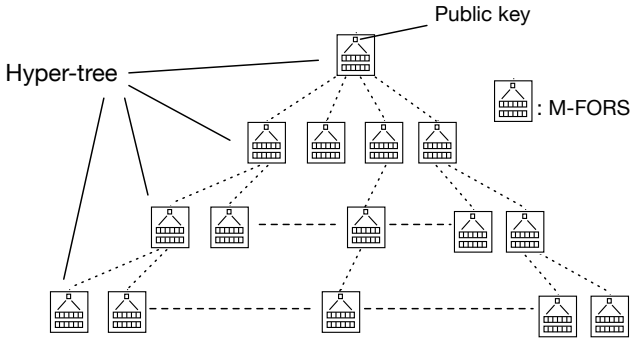

Figure 1: F-SPHINCS+ signatures.

2.3.1 M-FORS

The M-FORS signature scheme, as depicted in Fig. 2, is a modification of FORS used in SPHINCS+ [4]. As mentioned before, FORS is a few-time signature scheme such that each key pair can be used to sign up to q signatures. M-FORS, short for Merkle FORS, differs from FORS in that, the public key is generated as the root of a Merkle tree. The purpose of this modification is to allow a partial proof in MPCitH, which will significantly imrove the performance (see M-FORS partial proof later for a more detailed discussion). With M-FORS, the hyper-tree in F-SPHINCS+ is a q-ary tree such that the public key in a child

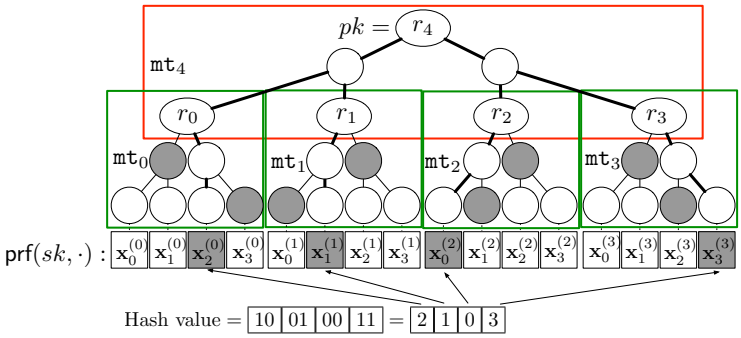

Figure 2: M-FORS signatures.

node is signed by the signing key in the parent node, and the signing key in the leaf node signs the actual message hash. An F-SPHINCS+ signature then contains a list of $h+1$ signatures, where h is the height of the hyper-tree. The benefit of M-FORS over XMSS that is used in the original SPHINCS+ scheme is the lower verification cost. To verify a message hash that is k blocks of d-bit string, the cost is $d \cdot k + k - 1$ hash operations. This is much less than the $(2^d - 1) \cdot k$ hashes for verifying a WOTS+ signature. On the other hand, the signing time is more than that of WOTS+. However, this is a lesser concern because in our case signing will be done in the clear (while verification needs to be done with zero knowledge). M-FORS consists of the algorithms below.

- $\mathsf{keyGen}(\mathtt{seed}, n, d, k, \mathtt{aux})$: it takes as input a random seed $\mathtt{seed}$, a security parameter n, two positive integers d and k, and $\mathtt{aux}$ that is either an empty string or some optional data. If $\mathtt{seed}$ is an empty string, an n-bit random string will be chosen and assigned to it. Then a pseudorandom function prf is used to expand $\mathtt{seed}$ into k lists $(\mathbf{x}^{(0)}, \cdots, \mathbf{x}^{(k-1)})$, where each $\mathbf{x}^{(i)}$ contains 2^d distinct n-bit pseudorandom strings. Then $k+1$ Merkle trees $\mathbf{T} = (\mathtt{mt}_0, \cdots, \mathtt{mt}_k)$ are built. In particular, each of $\mathtt{mt}_0, \cdots, \mathtt{mt}_{k-1}$ has 2^d leaf nodes. The jth leaf node in $\mathtt{mt}_i$ is the hash of $\mathbf{x}_j^{(i)}$. The leaf nodes of $\mathtt{mt}_k$ are $r_0, \cdots, r_{k-1}$ that are the roots of $(\mathtt{mt}_0, \cdots, \mathtt{mt}_{k-1})$. keyGen outputs $(pk, sk, param)$, such that the public key $pk = r_k$ where r_k is the root of $\mathtt{mt}_k$, the private key $sk = \mathtt{seed}$, and the public parameters $mp = (n, d, k, \mathtt{aux})$.

- $\mathsf{sign}(sk, MD, mp)$: to sign a message hash $MD \in \{0,1\}^{k \cdot d}$, parse it into k blocks, each block is interpreted as a d-bit unsigned integers $(p_0, \cdots, p_{k-1})$. Then for the i-th block p_i, $\mathbf{x}^{(i)}$ and $\mathtt{mt}_i$ (obtained by expanding sk) are used to generate $\mathtt{authpath}^{(i)}$, which is the authentication path of the p_i-th leaf node in the i-th Merkle tree. Then $(\mathbf{x}_{p_i}^{(i)}, \mathtt{authpath}^{(i)})$ is put into the signature. The signature is a list of k pairs $\boldsymbol{\sigma} = \{(\mathbf{x}_{p_0}^{(0)}, \mathtt{authpath}^{(0)}), \cdots, (\mathbf{x}_{p_{k-1}}^{(k-1)}, \mathtt{authpath}^{(k-1)})\}$.

- $\mathsf{recoverPK}(\sigma, MD, mp)$: This algorithm outputs the public key recovered from a signature σ and the message hash MD. First MD is parsed into k blocks

$(p'_0, \cdots, p'_{k-1})$. Then for $0 \leq i \leq k-1$, $\boldsymbol{\sigma}_i = (x_i, \texttt{authpath}^{(i)})$ and p'_i are used to re-generate a Merkle tree root and get the value r'_i (p'_i is used to determine the order of the siblings at each layer). Finally, $r'_0, \cdots, r'_{k-1}$ are used to compute $\texttt{mt}'_k$ and its root r'_k is returned.

- $\mathsf{verify}(\sigma, pk, MD, mp)$: to verify a signature, call $\mathsf{recoverPK}(\sigma, MD, mp)$. If the recovered public key is the same as pk, accept the signature, otherwise reject.

2.3.2 F-SPHINCS+

The hyper-tree nodes in F-SPHINCS+ are addressed by a pair (a, b) where a is its layer and b is its index within the layer. The root node is at layer 0, and the layer number of all other nodes is the layer number of its parent plus 1. All nodes within a layer are viewed as an ordered list, and index each node in the list from left to right, starting from 0. F-SPHINCS+ consists of the following algorithms:

- $\mathsf{keyGen}(n, q, h)$: This algorithm outputs (sk, pk, fp). It takes as input a security parameter n, the degree of non-leaf nodes in the hyper-tree q, and the height of the hyper-tree h. Then it chooses d, k that are the parameters for the underlying M-FORS signature scheme. The public parameters are $fp = (n, q, h, d, k)$. It also chooses an n-bit random string as the private key sk. It generates the M-FORS key pair for the root node by calling $\mathsf{genNode}((0,0), sk, fp)$, and set the public key pk to be the M-FORS public key $pk_{0,0}$.

- $\mathsf{genNode}(nodeAdr, sk, fp)$: This algorithm generates a node in the hyper-tree given the address $nodeAdr = (a, b)$. With the private key sk used as a *seed*, the algorithm first generates a subseed with a pseudorandom function $\texttt{seed}_{a,b} = \mathsf{prf}(seed, a||b)$, then it calls M-FORS key generation algorithm $\mathsf{M\text{-}FORS.keyGen}$ $(\texttt{seed}_{a,b}, n, d, k, a||b)$. The output $(pk_{a,b}, sk_{a,b}, mp_{a,b})$ is the content of the node at (a, b).

- $\mathsf{mHash}(msg, gr)$:This algorithm produces message hash and the leaf node index used in generating the F-SPHINCS+ signature. The input msg is the message to be signed, gr is a random string. The algorithm produces $MD||idx \leftarrow H_3(msg||gr)$, where $H_3 : \{0,1\}^* \rightarrow \{0,1\}^{d \cdot k + (\log_2 q) \cdot h}$ is a public hash function, MD is $d \cdot k$ bit long and idx is interpreted as an $(\log_2 q) \cdot h$ bit long unsigned integer.

- $\mathsf{sign}(msg, sk, fp)$: This algorithm produces the F-SPHINCS+ signature as a chain of M-FORS signature along the path from a leaf node to the root node of the hyper-tree. It chooses an n-bit random string gr. Then obtain $MD||idx \leftarrow \mathsf{mHash}(msg, gr)$. A leaf node at (h, idx) is then generated by calling $\mathsf{genNode}((h, idx), sk, fp)$. The M-FORS signing key $sk_{h,idx}$ is used to sign MD and generate σ_0. The parent node of (h, idx) is then generated by calling $\mathsf{genNode}((h-1, b), sk, fp)$ where $(h-1, b)$ is the address of the parent

node. Then the parent secret key $sk_{h-1,b}$ is used to sign the child public key $pk_{h,idx}$, and the signature is σ_1. Repeat the signing process until obtaining σ_h that is signed by $sk_{0,0}$ on $pk_{1,b'}$ for some b'. The F-SPHINCS+ signature is then $(gr, \mathbf{S} = (\sigma_0, \cdots, \sigma_h))$.

- $\mathsf{verify}(msg, gr, \mathbf{S}, pk, fp)$: This algorithm verifies every M-FORS signature that is chained up in $\mathbf{S}$. Given $\mathbf{S} = (\sigma_0, \cdots, \sigma_h)$, first compute $MD||idx \leftarrow H_3(msg||gr)$. Then obtain $pk_0 \leftarrow \mathsf{recoverPK}(\sigma_0, MD, mp_0)$, $pk_1 \leftarrow \mathsf{recoverPK}(\sigma_1, pk_0, mp_1)$, repeat until $pk_h \leftarrow \mathsf{recoverPK}(\sigma_h, pk_{h-1}, mp_h)$. If $pk = pk_h$, accept the signature, otherwise reject.

Remark 1 In M-FORS algorithms, we use two tweakable hash functions [4] $H_1 : \{0,1\}^* \to \{0,1\}^n$ and $H_2 : \{0,1\}^* \to \{0,1\}^{d \cdot k}$. Almost all hash operations are done using H_1. H_2 is only used to map the k-th Merkle tree to the $k \cdot d$-bit M-FORS public key, so that when used in F-SPHINCS+ the public key is of the right size to be signed by the parent node. If M-FORS is to be used as a stand-alone signature scheme, these two hash functions can be the same.

Remark 2 The tweakable hash functions follow Construction 7 for tweakable hash functions in [4]. Namely, the hash of an input M is produced by calling a hash function with additional input as $H(\mathtt{P}||\mathtt{ADD}||M)$, where $\mathtt{P}$ is a public hash key and $\mathtt{ADD}$ acts as the tweak. The tweak is the address where the hash operation takes place within the hyper-tree, and it is a five part string $a_1||b_1||v||a_2||b_2$:

- (a_1, b_1), where $0 \leq a_1 \leq h, 0 \leq b_1 \leq 2^{a_1} - 1$, is the address of an hyper-tree node. Within the node, an M-FORS key pair that is based on $k + 1$ Merkle trees are stored.

- $0 \leq v \leq k$ is the index of a Merkle tree in the M-FORS key pair stored in the hyper-tree node (a_1, b_1). When $0 \leq v \leq k - 1$, the Merkle tree (of height d) is used to sign the v-th block of the message; when $v = k$, the Merkle tree (of height $\lceil \log_2 k \rceil$) is used to accumulated the roots of all the previous Merkle trees into the public key.

- (a_2, b_2) is the address of an Merkle tree node. When $0 \leq v \leq k-1$, $0 \leq a_2 \leq d$ and $0 \leq b_2 \leq 2^{a_2} - 1$; When $v = k$, $0 \leq a_2 \leq \lceil \log_2 k \rceil - 1$ and $0 \leq b_2 \leq 2^{a_2} - 1$.

The security analysis of F-SPHINCS+ is given in Appendix A.

2.3.3 M-FORS partial proof

The challenge for implementing a NIZKP of F-SPHINCS+ signature with MPCitH comes from the cost of $h+1$ M-FORS signature verifications. Following the description of M-FORS, to verify a single M-FORS signature, $k \cdot (d+1) + (k-1) = kd + 2k - 1$ hashes are needed, which is in the order of 100 for a practical setting (with an extra factor of 2 if implementing with $\mathsf{MPC\_F}$, as a part of our EPID scheme, which will be described in Section 3). The $h + 1$ factor means that if implemented naively, the MPC would need to call thousands of times the sub-procedure that implements the hash function, and the size of the circuit

for the whole MPC can go easily above a million-gates. Even worse, to reduce the soundness error, the same circuit needs to be executed tens to hundreds of times in an MPCitH proof. Thus, a naive implementation of a NIZKP of F-SPHINCS+ signature will result in a very large signature size and a high computational cost.

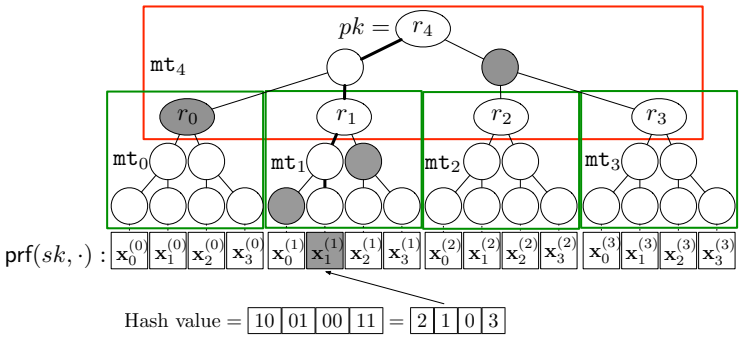

Figure 3: M-FORS Patial Verification.

As suggested in [21, 22], in an efficient MPCitH proof, rather than repeating t times an MPC procedure in which the M-FORS signatures are fully verified, we run $t' \geq k$ MPC procedures in which the M-FORS signatures are partially verified, one block in each run (see the example of partial verification in Figure 3). More precisely, we extend the M-FORS with the following algorithms:

- $\mathsf{partialSig}(\sigma, MD, i, mp)$: to extract a partial signature of the i-th block of MD from $\sigma = \{(x_0, \mathbf{authpath}^{(0)}), \cdots, (x_{k-1}, \mathbf{authpath}^{(k-1)})\}$. The Merkle tree $\mathtt{mt}_k$ can be recomputed from σ. The partial signature is $\partial_{\sigma,i} = (x_i, \mathbf{authpath}^{(i)}, \mathbf{authpath}^{(k,i)})$ where $(x_i, \mathbf{authpath}^{(i)})$ is a copy of the i-th pair in σ, and $\mathbf{authpath}^{(k,i)}$ is the authentication path of r_i (the root of the i-th Merkle tree) in $\mathtt{mt}_k$.
- $\mathsf{partialRec}(\partial_{\sigma,i}, p_i, i, mp)$: This algorithm recovers the public key from $\partial_{\sigma,i}$ and p_i. Given $\partial_{\sigma,i} = (x, \mathbf{authpath}, \mathbf{authpath}')$, first compute the Merkle tree root r_i from $(x, \mathbf{authpath}, p_i)$, then compute the Merkle tree root pk from $(r_i, \mathbf{authpath}', i)$. Output pk.

With partial-rec, only one path is used to recover the M-FORS public key instead of k paths. Why does this approach make sense? In an MPCitH proof, the same procedure is run multiple times. Each run has a soundness ϵ that a cheating prover can get away without being detected. Thus t runs are needed so that ϵ^t is negligibly small. In our case, the main cost of the MPC procedure comes from verifying all the M-FORS signatures. The full verification requires every block of the message digest or the child public key to be verified. Our observation is that if a prover has to cheat, then it has to cheat in more than 1 blocks with a high probability. If the prover has to cheat in n out of k

blocks, then using partial verification with t', such that $t' \cdot n/k \geq t$, ensures that the prover has to cheat in more than t runs, and hence with a negligible success probability. As we analysed, an implementation with full signature verification requires $t \cdot (h+1) \cdot (k \cdot d + 2k - 1)$ calls to the MPC hash procedure. The partial verification based implementation, on the other hand, requires only $t' \cdot (h+1) \cdot (d + 1 + \lceil \log k \rceil)$ MPC hash calls. The improvement is roughly $\frac{tk}{t'}$ times.

3 The Proposed EPID Scheme

3.1 Syntax

Players. The EPID scheme involves the following players:

- **An issuer** owns a group public and private key pair and manages the group membership, decides who can be a group member, and issues a group membership credential to each group member. Each group has one issuer and the issuer is also called the group manager.
- **A group member** generates a public and private key pair, receives a membership credential from the issuer for this key pair, and uses the key and credential to create EPID signatures. A group member is also called an EPID signer. Each group has many members.
- **A verifier** verifies EPID signatures by using the issuer's group public key. Each group can have an arbitrary number of verifiers.
- **A revocation authority** decides whether a group member should be removed from the group and maintains signature and key revocation lists. Each group should have at least one revocation authority.

Revocation lists. There are two types of revocation lists:

- A key revocation list denoted by `KRL` lists revealed private signing keys.
- A signature revocation list denoted by `SRL` lists signatures created by revoked signers.

Each verifier uses one `KRL` and one `SRL`, from their chosen revocation authority. Note that key revocation provides the ability to revoke an existing signature with an updated key revocation list. However, as for any EPID scheme, signature revocation does not provide such ability because a signature can only prove that the signing key is not associated with any key in the `SRL` that is available during signing. This proof cannot be extended to cover any items added into the `SRL` later.

Algorithms/protocols. Our EPID scheme consists of the following algorithms/protocols:

- $\mathsf{Init}(n)$: In the initialization algorithm, the issuer takes a security parameter n as the input, and outputs a master (group) key pair $(mpk,\ msk)$. The master public key mpk is made public and the master secret key msk is stored privately by the issuer. In all other algorithms and protocols, we will assume mpk along with the security parameter n as an implicit input for all parties.

- $\mathsf{Join}(msk)$: the joining protocol is an interactive protocol between the issuer and a user u who wants to join the group. The issuer has a private input msk and the user does not have input. At the end of the protocol, the issuer outputs a decision: `accept` or `reject`. If `reject`, then stop. If `accept`, the user obtains its group signing key $gsk_u = (sk_u,\ cred_u)$ where sk_u is a secret key, and $cred_u$ is a group membership credential. sk_u is chosen and held by the user, and $cred_u$ is generated by the issuer and given to the user. Now, the user becomes a group member, who is a legitimate signer.

- $\mathsf{Sign}(gsk_u,\ msg,\ \mathtt{SRL})$: the signing algorithm allows a signer to produce an EPID signature Σ on a message $msg \in \{0,1\}^*$ using its signing key gsk_u. The signature includes proof of whether gsk_u was used to generate any signatures in the signature revocation list `SRL`.

- $\mathsf{Verify}(msg, \Sigma, \mathtt{KRL}, \mathtt{SRL})$: the verification algorithm allows a verifier to verify whether a signature Σ is a valid signature of msg, whether the corresponding signing key has been listed on a rogue key list `KRL`, and whether it has been used to generate any signatures on the signature revocation list `SRL`.

- $\mathsf{Revocation}(\mathtt{KRL},\mathtt{SRL})$: The revocation algorithm allows a revocation authority to add a revealed signing key in `KRL` and to add a signature generated by a revoked signer in `SRL`.

3.2 Details of algorithms and protocols

Initialization $\mathsf{Init}(n)$: Given a security parameter n, the issuer does the following to create a group master public and secret key pair to be used for the F-SPHINCS+ signature scheme.

1. The issuer first chooses the hyper-tree node degree q and the tree height h, the values (d, k) for the underlying M-FORS scheme, a pseudorandom function prf, three hash functions $H_1 : \{0,1\}^* \to \{0,1\}^n$, $H_2 : \{0,1\}^* \to \{0,1\}^{d \cdot k}$, $H_3 : \{0,1\}^* \to \{0,1\}^{d \cdot k + (\log_2 q) \cdot h}$, and a keyed pseudorandom function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$.

2. The issuer then runs $(sk, rpk, gp) \leftarrow \mathsf{F\text{-}SPHINCS+.keyGen}(n, q, h)$, where (rpk, sk) is the F-SPHINCS+ public and secret key pair, and $gp = (n, q, h, d, k)$ are the hyper-tree parameters.

3. The issuer then publishes the master public key $mpk = (gp,\ rpk,\ H_1,\ H_2,\ H_3,\ F,\ \mathsf{prf})$ and keeps the master secret key $msk = sk$ private.

4. To prove the key construction correctness, the issuer provides a Non-Interactive Zero-Knowledge Proof (NIZKP) $\pi_{\mathcal{I}}$ to demonstrate that the key pair is generated correctly, meaning that the secret and public keys are associated with each other. This NIZKP can be achieved by signing its own public key rpk using F-SPHINCS+.sign, which is similar to the issuer creating a group membership credential in the joining protocol described below.

5. In addition, the issuer initializes an empty group list GL.

Each verifier initializes two revocation lists: a key revocation list KRL and a signature revocation list SRL. All these lists are empty when initialised. Alternatively, these two revocation lists can be managed by a centralised authority and each signer and verifier can then download them for signing or verification.

Joining protocol $\mathsf{Join}(msk, n)$: The joining protocol is run between a user and the issuer. Note that this protocol involves the authentication of the user by the issuer. The issuer has an authentic copy of the user's endorsement key, which is used to establish a secure and authenticated channel between the user and the issuer. Our EPID scheme does not restrict which cryptographic mechanism is used to establish this channel. In the following protocol description, it is assumed the existence of such a channel. The protocol includes the following steps:

1. A unique session ID u is assigned to the user. For simplicity, we can think of the session ID as a monotonically increasing counter, and each invocation of the joining protocol will increase it by 1. Alternatively, the value u can be computed from the user's endorsement key, which is unique to the user.

2. The user chooses a random secret key: $sk_u \xleftarrow{R} \{0,1\}^n$ as its signing key.

3. The user computes the group identifier $gid = H_1(rpk)$.

4. The user generates its entry token $et_u = F(sk_u, gid)$ together with the NIZKP $\pi_u : \mathcal{P}\{(gp,\ gid,\ et_u); (sk_u) | et_u = F(sk_u,\ gid)\}$.

5. The user chooses a random string $cr \xleftarrow{R} \{0,1\}^n$, computes a commitment $ct = H_1(et_u||cr)$, then sends (u, ct) to the issuer to request joining the group.

6. Upon receiving (u, ct), the issuer checks whether an entry with the same u is in GL. If yes, the issuer rejects the user. Otherwise, if the issuer would like to accept the user, the issuer chooses a random string $gr_u \xleftarrow{R} \{0,1\}^n$ and sends it to the user, who responds by sending (et_u, cr, π_u) back. The issuer verifies $ct = H_1(et_u||cr)$ and the NIZKP π_u. If both verifications pass, the issuer computes the group credential $(gr_u, \mathbf{S}) \leftarrow \mathsf{F\text{-}SPHINCS+.sign}(et_u||gr_u, msk, gp)$ and adds $(u,\ et_u,\ gr_u, \mathbf{S})$ to GL; otherwise, the issuer rejects the user.

7. The user, if accepted by the issuer, sets its group signing key $gsk_u = (sk_u,\ gr_u, \mathbf{S})$. The user has now become an EPID signer for this group.

Signature generation $\mathsf{Sign}(gsk_u, msg, \mathtt{SRL})$: Given a message to be signed denoted by msg and a signature revocation list $\mathtt{SRL}$, the user u generates an EPID signature using $gsk_u = (sk_u,\ gr_u,\ \mathbf{S})$ by performing the following steps:

1. The user gets the group public key rpk and generates a random data string $str \xleftarrow{R} \{0,1\}^n$, then computes sid, sst, gid, et_u, mt_u, idx as follows:
   (a) The user first computes the signature identifier $sid = H_1(msg||str)$ and the signature signing token $sst = F(sk_u, sid)$.
   (b) The user then computes the group identifier $gid = H_1(rpk)$, the group membership entry token $et_u = F(sk_u, gid)$ and $mt_u||idx = H_3(et_u||gr_u)$. Here $H_3(et_u||gr_u)$ is used as $\mathsf{F\text{-}SPHINCS{+}.mHash}(et_u, gr_u)$.
2. To prove that the user has not been revoked, the user generates a random number $r \xleftarrow{R} \{0,1\}^n$, then $\forall j \in \{1, \ldots, J = |\mathtt{SRL}|\}$, the user retrieves $sid_j = H_1(msg_j||str_j)$ and $sst_j = F(sk_j, sid_j)$ from $\mathtt{SRL}$, and produces two signature proof tokens $A_j = F(F(sk_u, sid_j), r)$ and $B_j = F(sst_j, r)$. Note that if and only if $sk_u \neq sk_j$, $A_j \neq B_j$. Also note that $F(sk_u, sid_j)$ has been kept secret, so that (A_j, B_j) will not reveal any sensitive information about sk_u. As discussed in Remark 3 below, **this novel and efficient privacy-preserving inequivalence proof may have its interest.**
3. To prove the EPID credential, the user generates another random number $s \xleftarrow{R} \{0,1\}^n$ and computes $com = H_1(s||pk_h|| \cdots ||rpk)\}$, where $pk_h, \cdots, rpk$ are the public keys for verifying the signatures in $\mathbf{S}$, from the layer h to layer 0 (the public key at the layer 0 is rpk). The user produces an NIZKP $\pi_{\mathtt{E}}$ (note that $\mathsf{recoverPK}$ was defined in Subsection 2.3.1 for M-FORS):

$$
\begin{aligned}
\pi_{\mathtt{E}} :& \mathcal{P}\{(gp,\ rpk,\ gid,\ sid,\ sst,\ com,\ r,\ \mathtt{SRL}, \forall_{j\in[1,J]}\ A_j);\\
& (sk_u,\ et_u,\ gr_u,\ s,\ \mathbf{S} = \{\sigma_h, \cdots, \sigma_0\}) \mid sst = F(sk_u,\ sid)\\
& \wedge \forall_{j\in[1,J]}\ A_j = F(F(sk_u,\ sid_j), r)\\
& \wedge et_u = F(sk_u,\ gid)\\
& \wedge mt_u||idx = H_3(et_u||gr_u)\\
& \wedge pk_h = \mathsf{recoverPK}(\sigma_h, mt_u, (n, d, k, (h, idx)))\\
& \wedge pk_{h-1} = \mathsf{recoverPK}(\sigma_{h-1}, pk_h, (n, d, k, (h-1, \lfloor \frac{idx}{q} \rfloor))) \wedge \cdots\\
& \wedge rpk = \mathsf{recoverPK}(\sigma_0, pk_1, (n, d, k, (0, 0)))\\
& \wedge com = H_1(s||pk_h|| \cdots ||rpk)\}
\end{aligned}
$$

4. The EPID signature is $\Sigma = (str,\ sst,\ com,\ r,\ \forall_{j\in[1,J]}\ A_j, \pi_{\mathtt{E}})$.

Signature verification $\mathsf{Verify}(msg,\ \Sigma,\ \mathtt{KRL},\ \mathtt{SRL})$: Given $\Sigma = (str, sst, com,\ r,\ \forall_{j\in[1,J]}\ A_j, \pi_{\mathtt{E}})$ and msg together with two revocation lists $\mathtt{KRL}$ and $\mathtt{SRL}$, the verifier performs the following steps to verify this signature:

1. $\forall sk_j \in \mathtt{KRL}$, the verifier computes $sst_j^* = F(sk_j, sid)$. If any $sst_j^* = sst$, rejects Σ.

2. Otherwise, $\forall j \in [1, J]$, the verifier checks if any sid_j has been used in Σ correctly. If not reject Σ. Otherwise, compute $B_j = F(sst_j, r)$ and check if any $A_j = B_j$ holds, if so reject Σ.

3. Otherwise, the verifier recomputes $sid = H_1(msg||str)$ and $gid = H_1(rpk)$, then verifies $\pi_{\mathtt{E}}$. Accept if the verification succeeds, otherwise reject.

Key and signature revocation Revocation(KRL,SRL) : There are two cases to revoke the group membership of the user u:

1. Given sk_u, the revocation authority adds it in KRL.

2. Given a signature Σ signed by the user u together with the corresponding signed message msg, the revocation authority retrieves str and sst from Σ, computes $sid = H_1(msg||str)$, and adds the pair (sid, sst) in SRL.

Remark 3 In the existing EPID scheme [8], an EPID signature under the signing key sk_i contains of $t = (F(sk_i, r), r)$, where r is a random value. To support signature revocation, for each revoked signature $\mathsf{sig}_j \in \mathtt{SRL}$, a new EPID signature under the signing key sk_i^* includes the NIZKP for $t_{\mathsf{sig}_j} \neq (F(sk_i^*, r_{\mathsf{sig}_j}), r_{\mathsf{sig}_j})$. However, the paper does not provide a concrete construction for this inequality proof in zero-knowledge. To build a practical EPID scheme from symmetric primitives, a concrete proof scheme is necessary. For the first time, we propose a non-interactive zero-knowledge inequality proof $\pi_{\mathtt{R}}$:

$$\pi_{\mathtt{R}} : \mathcal{P}\{(gp,\ sid,\ sst,\ r,\ \forall_{j\in[1,J]} : (sid_j,\ sst_j) \in \mathtt{SRL},\ A_j); (sk_u)|$$
$$sst = F(sk_u,\ sid) \wedge \forall_{j\in[1,J]}\ A_j = F(F(sk_u,\ sid_j), r)\}$$

where sk_u is the user's long-term key and r is a public nonce, which allows the verifier to compute $B_j = F(sst_j, r)$. If $A_j \neq B_j$, the verifier can be convinced that $\mathsf{sig}_j \in \mathtt{SRL}$ was not signed under sk_u; since otherwise it contradicts the collision-resistance of the keyed pseudorandom function F. However, if $sk_u = sk_j$, $A_j = B_j$ must hold. This proof is privacy-preserving for an unrevoked user since such proof does not reveal any sensitive information of sk_u. The proof is reasonably efficient. When SRL has J revoked signatures, it requires the NIZKP for $2J$ operations of F. In § 3.5, we will discuss how to separate this revocation proof from the credential proof to optimise the performance of our EPID scheme.

3.3 The proof $\pi_{\mathtt{E}}$

The most important part in the EPID signature $\Sigma = (str, sst, com, r, \forall_{j\in[1,J]}\ A_j, \pi_{\mathtt{E}})$ is the proof $\pi_{\mathtt{E}}$. As Σ is a signature of a message msg, the foremost thing $\pi_{\mathtt{E}}$ needs to prove is that the signer knows a group signing key $gsk_u = (sk_u,\ gr_u,\ \mathbf{S})$ and it was used to sign msg. Besides that, $\pi_{\mathtt{E}}$ also needs to prove that gsk_u is authorized by the issuer. To do that, in $\pi_{\mathtt{E}}$ the following is done:

1. It proves that given a nonce r, $\forall_{j\in[1,J]}\ sid_j \in \mathtt{SRL}$, $A_j = F(F(sk_u, sid_j), r)$ is correctly computed.

2. It proves that the same signing key sk_u used in the previous step is also used to generate two values et_u and sst, where et_u is bound with the group root public key rpk (as it is computed from $gid = H_1(rpk)$) and sst is bound with the message msg and random string str (as it is computed from $sid = H_1(msg||str)$). sst is revealed in Σ, and et_u is hidden.

3. It proves that mt_u, which is computed from et_u, is signed under a private key in a leaf node of the hyper-tree generated by the group issuer. This is done by verifying all the signatures in $\mathbf{S}$ such that mt_u and σ_h produce the leaf public key pk_h, which in turn with σ_{h-1} produces pk_{h-1}, and so on until reaching the root. The last public key produced is rpk which is published by the group issuer. All public keys recovered in this process match those committed in the commitment com.

Before showing the MPCitH instance of $\pi_{\mathtt{E}}$, let us first introduce the notation used in such an MPCitH algorithm: $[\![x]\!]$ means that the value x is secret-shared when using an MPC algorithm, meaning that x is known by the prover but not the verifier. MPC_X means the MPC subroutine implementing the function X (e.g. MPC_F, MPC_H1, MPC_H2 and MPC_H3 implement F, H_1, H_2 and H_3). This notation will be used throughout the paper. Based on [36], in an implementation MPC_F can be used as a building block for the hash functions that we need.

Following the method of M-FORS partial verification, as shown in Subsection 2.3.3, we now introduce the $\pi_{\mathtt{E}}$ MPC instance for the v-th block in MPCitH 1. The user uses partial signatures in the MPC. Recall that in the group signing key gsk_u, a list $\mathbf{S} = \{\sigma_h, \cdots, \sigma_0\}$ of $h+1$ signatures are stored, one for each layer in the hyper-tree of F-SPHINCS+. The signer can extract a partial signature for the v-th block from each signature, i.e. $\{\partial_{\sigma_h,v}, \cdots, \partial_{\sigma_0,v}\}$. In Line 12, an MPC subroutine MPC_partialRec that implements partialRec is used. This subroutine uses the input to compute the corresponding public key at the l-th layer in the hyper-tree (stored in $[\![M]\!]$ and also appended to $[\![COM]\!]$). After the last iteration, $[\![COM]\!]$ is hashed and $[\![M]\!]$ is revealed. The results will be checked by the verifier to see whether they match com and rpk. If so, the signer is likely to possess valid partial signatures along the path from the idx-th leaf node to the root node in the hyper-tree. Note that the algorithm partialRec was defined in Subsection 2.3.3 for M-FORS partial proofs and the algorithm Reveal($[\![x]\!]$) is simply unmasking the value x.

3.4 Soundness Analysis of $\pi_{\mathtt{E}}$

In $\pi_{\mathtt{E}}$, k instances of MPC are run. In the ith instance, the partial verification procedure is used to verify every M-FORS signature in $\mathbf{S}$, but only the i-th block of the hash value being signed. Out of the k blocks, the adversary may have learned the secret strings correspond to λ_1 blocks through queries and has to cheat in all the remaining $k - \lambda_1$ blocks. For each MPC instance, the verifier opens the views of a subset of the MPC parties and a cheat prover can be

MPCitH 1: $\pi_{\mathtt{E},v}$ – MPC instance for the v-th block

Public: $gp = (n,\ q,\ h,\ d,\ k)$,
$rpk,\ gid,\ sid,\ sst,\ com,\ r,\ v,\ \forall_{j\in[1,J]}\ (sid_j, A_j)$
Private: $[\![sk_u]\!],\ [\![etu]\!],\ [\![gr_u]\!],\ [\![s]\!],\ [\![\partial_{\sigma_h,\ v}]\!],\ \cdots,\ [\![\partial_{\sigma_0,\ v}]\!]$
Output: $pk_0,\ \forall_{j\in[1,J]}\ A'_j,\ com'$
Check: $pk_0 = rpk\ \wedge\ com' = com\ \wedge\ \forall_{j\in[1,J]}\ (A'_j = A_j)$

for $j = 1;\ j \le |\mathtt{SRL}|;\ j{+}{+}$ **do**
  $[\![A]\!] = \mathsf{MPC\_F}([\![sk_u]\!], sid_j)$;
  $A'_j = \mathsf{MPC\_F}([\![A]\!], r)$;
end
$sst = \mathsf{MPC\_F}([\![sk_u]\!],\ sid)$;
$[\![et_u]\!] = \mathsf{MPC\_F}([\![sk_u]\!],\ gid)$;
$[\![mt_u]\!]||[\![idx]\!] = \mathsf{MPC\_H3}([\![et_u]\!]||[\![gr_u]\!])$;
$[\![M]\!] = [\![mt_u]\!]$;
$[\![COM]\!] = [\![s]\!]$;
for $l = h;\ l \ge 0;\ l{-}{-}$ **do**
  parse $[\![M]\!]$ into k blocks $[\![p_0]\!], \cdots, [\![p_{k-1}]\!]$, each block is d-bits;
  $[\![M]\!] = \mathsf{MPC\_partialRec}([\![\partial_{\sigma_l,v}]\!],\ [\![p_v]\!],\ [\![idx]\!],\ gp,\ l,\ v)$;
  $[\![COM]\!] = \mathsf{MPC\_H1}([\![COM]\!]||[\![M]\!])$;
  $[\![idx]\!] = [\![\lfloor idx/q \rfloor]\!]$;
end
$com' = \mathsf{Reveal}([\![COM]\!])$;
$pk_0 = \mathsf{Reveal}([\![M]\!])$;

detected with a probability $1-\epsilon$. Therefore, if using an MPC protocol without pre-processing, then the soundness error is;

$$\sum_{i=0}^{k} \Pr[\lambda_1 = i] \cdot \epsilon^{k-i}$$

If using an MPC protocol with pre-processing, then the adversary can also cheat in the pre-processing phase. If the adversary cheats in λ_2 (out of M) copies of pre-processing data, and not being detected when checking the pre-processing data (the probability is denoted as $\mathrm{Succ}^{pre}(\lambda_2, k, M)$), then it needs to cheat in $k - \lambda_1 - \lambda_2$ MPC instances. The soundness error is:

$$\sum_{i=0}^{k} \Pr[\lambda_1 = i] \left(\sum_{\lambda_2=0}^{k-\lambda_1} \mathrm{Succ}^{pre}(\lambda_2, k, M) \cdot \epsilon^{k-\lambda_1-\lambda_2} \right)$$

As a concrete example, let us consider a case in which we implement $\pi_{\mathtt{E}}$ using KKW [39]. Then we have:

$$\Pr[\lambda_1 = i] = \binom{k}{i} (1 - (1-2^{-d})^q)^i ((1-2^{-d})^q)^{k-i},$$

$$\mathrm{Succ}^{pre}(\lambda_2, k, M) = \frac{\binom{M-\lambda_2}{M-k}}{\binom{M}{M-k}}, \quad \epsilon = \frac{1}{N}$$

In the above, d, k, q are the parameters for the M-FORS signature, M is the number of pre-processing data generated, and N is the number of MPC parties. When $d = 16, k = 70, q = 1024, M = 1120$, and $N = 16$, then the soundness error is $2^{-257.769}$; when $d = 16, k = 35, q = 1024, M = 560$, and $N = 16$, then the soundness error is $2^{-128.987}$.

3.5 Splitting revocation and credential proofs

The cost of proving or verifying that a signer is not revoked based on a given SRL is dependent on the size of SRL, while the cost of proving or verifying that a signer holds a valid group credential is related to the group size. The previous specification of $\pi_{\mathtt{E}}$ in Subsection 3.3 includes these two proofs in a single NIZKP. This may not be the best choice if the costs of these two proofs are not balanced, so optionally we split the proof $\pi_{\mathtt{E}}$ into two separate NIZKPs, one for revocation denoted by $\pi_{\mathtt{R}}$ and another for the credential denoted by $\pi_{\mathtt{C}}$, i.e., $\pi_{\mathtt{E}} = (\pi_{\mathtt{R}}, \pi_{\mathtt{C}})$. We specify them in MPCitH 2 and MPCitH 3 respectively. Note that $\pi_{\mathtt{R}}$ does not use a partial proof but $\pi_{\mathtt{C}}$ does. The connection between these two NIZKPs is that they share the same sid and sst values, which indicates that these two NIZKPs are created by the same sk_u.

MPCitH 2: $\pi_{\mathtt{R}}$ – MPC Instance for Revocation

Public: $gp = (n,\ q,\ h,\ d,\ k),\ sid,\ sst,\ r,\ \forall_{j\in[1,J]}\ (sid_j, A_j)$
Private: $[\![sk_u]\!]$
Output: $sst',\ \forall_{j\in[1,J]}\ A'_j$
Check: $sst = sst' \ \wedge\ \forall_{j\in[1,J]}\ A'_j = A_j$

$sst' = \mathsf{MPC\_F}([\![sk_u]\!],\ sid)$;
for $j = 1;\ j \le |\mathtt{SRL}|;\ j{+}{+}$ **do**
  $[\![A]\!] = \mathsf{MPC\_F}([\![sk_u]\!], sid_j)$;
  $A'_j = \mathsf{MPC\_F}([\![A]\!], r)$;
end

4 UC-based EPID Security Model

In this section, we recall the definition of the EPID UC model from [30], which is a modification of the DAA UC model given by Camenisch et al. in [17]. The changes include replacing linkability with a revocation interface and adding the signature revocation check from [16]. The model covers the following security properties:

Correctness: If a group member has completed the Join procedure and neither its key nor any of its signatures have been revoked, that group member's

MPCitH 3: $\pi_{\mathsf{C},v}$ – MPC instance for the v-th block

Public: $gp = (n,\ q,\ h,\ d,\ k),\ rpk,\ gid,\ sid,\ sst,\ com,\ v$
Private: $[\![sk_u]\!],\ [\![etu]\!],\ [\![gr_u]\!],\ [\![s]\!],\ [\![\partial_{\sigma_h,\ v}]\!],\ \cdots,\ [\![\partial_{\sigma_0,\ v}]\!]$
Output: $pk_0,\ com', sst'$
Check: $pk_0 = rpk\ \wedge\ com' = com\ \wedge\ sst' = sst$

$sst' = \mathsf{MPC\_F}([\![sk_u]\!],\ sid)$;
$[\![et_u]\!] = \mathsf{MPC\_F}([\![sk_u]\!],\ gid)$;
$[\![mt_u]\!]||[\![idx]\!] = \mathsf{MPC\_H3}([\![et_u]\!]||[\![gr_u]\!])$;
$[\![M]\!] = [\![mt_u]\!]$;
$[\![COM]\!] = [\![s]\!]$;
for $\underline{l = h;\ l \geq 0;\ l{-}{-}}$ **do**
  parse $[\![M]\!]$ into k blocks $[\![p_0]\!], \cdots, [\![p_{k-1}]\!]$, each block is d-bit;
  $[\![M]\!] = \mathsf{MPC\_partialRec}([\![\partial_{\sigma_l,v}]\!],\ [\![p_v]\!],\ [\![idx]\!],\ gp,\ l,\ v)$;
  $[\![COM]\!] = \mathsf{MPC\_H1}([\![COM]\!]||[\![M]\!])$;
  $[\![idx]\!] = [\![\lfloor idx/q \rfloor]\!]$;
end
$com' = \mathsf{Reveal}([\![COM]\!])$;
$pk_0 = \mathsf{Reveal}([\![M]\!])$;

signatures should successfully verify.

Anonymity: Given two signatures, no adversary can distinguish whether they were created by one honest signer or two different honest signers.

Unforgeability: When the issuer is honest, no adversary can create a signature on behalf of an honest signer.

Non-frameability: Regardless of whether the issuer is honest or not, no adversary can create a signature that, if put in `SRL`, can revoke any honest signers' signatures.

Generally speaking, security in the UC framework follows the simulation-based paradigm, where a protocol is secure when it is as secure as an ideal functionality that performs the desired tasks in a way that is secure by design. In a UC model, an environment $\mathcal{E}$ passes inputs and outputs to the protocol parties. The network is controlled by an adversary $\mathcal{A}$ that may communicate freely with $\mathcal{E}$. In the ideal world, the parties forward their inputs to the ideal functionality $\mathcal{F}$, which then (internally) performs the defined task and creates outputs that the parties forward to $\mathcal{E}$. A real-world protocol Π is said to securely realize a functionality $\mathcal{F}$, if the real world is indistinguishable from the ideal world, meaning for every adversary performing an attack in the real world, there is an ideal world adversary (often called simulator) $\mathcal{S}$ that performs the same attack in the ideal world. More precisely, a protocol Π is secure if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{E}$ can distinguish executing the real world with Π and $\mathcal{A}$, and executing the ideal world with $\mathcal{F}$ and $\mathcal{S}$.

The ideal functionality $\mathcal{F}_{\mathsf{EPID}}$ is formally defined under the assumption of static corruption, i.e., the adversary decides beforehand which parties are corrupt and informs $\mathcal{F}_{\mathsf{EPID}}$ about them. $\mathcal{F}_{\mathsf{EPID}}$ has five interfaces (SETUP, JOIN, SIGN, VERIFY, REVOKE) described below. Several sessions of the protocol are allowed to run at the same time and each session will be given a global identifier $\mathtt{SID}$ that consists of an issuer $\mathcal{I}$ and a unique string $\mathtt{SID}'$, i.e. $\mathtt{SID} = (\mathcal{I}, \mathtt{SID}')$. We also define the JOIN and SIGN sub-sessions by $\mathtt{JSID}$ and $\mathtt{SSID}$. We also define the algorithms that will be used inside the functionality as follows:

- Kgen(1^λ): A probabilistic algorithm that takes a security parameter λ and generates keys tsk for honest signers.
- sig(tsk, msg): A probabilistic algorithm used for honest signers. On input of a key tsk, it calculates the signature identifier $sid = H_1(msg, str)$ for a random $str \leftarrow \{0,1\}^n$ and outputs a signature Σ.
- ver$(\Sigma, msg, \mathtt{KRL}, \mathtt{SRL})$: A deterministic algorithm that is used in the VERIFY interface. On input of a signature Σ, it outputs $f = 1$ if the signature is valid, $f = 0$ otherwise.
- $\mathsf{Identify}(tsk, \Sigma,\ msg)$: A deterministic algorithm that will be used to ensure consistency with the ideal functionality $\mathcal{F}_{\mathsf{EPID}}$'s internal records. It outputs 1 if a key tsk was used to produce a signature Σ on sid, 0 otherwise.
- Revoke$(tsk^*, \Sigma^*, \mathtt{KRL}, \mathtt{SRL})$: A deterministic algorithm that takes input tsk^* or Σ^*, it adds tsk^* to $\mathtt{KRL}$ or Σ^* to $\mathtt{SRL}$ respectively.

The UC framework allows us to focus on the analysis of a single protocol instance with a globally unique session identifier $\mathtt{SID}$. $\mathcal{F}_{\mathsf{EPID}}$ uses session identifiers of the form $\mathtt{SID} = (\mathcal{I}, \mathtt{SID}')$ for some issuer $\mathcal{I}$ and a unique string $\mathtt{SID}'$. In the procedures, functions $\mathsf{CheckTtdHonest}$ and $\mathsf{CheckTtdCorrupt}$ are used that return '1' when a key belongs to a honest signer that has produced no signature, and when a key belongs to a corrupt user such that there is no signature simultaneously linking back to the inputted key and another one, respectively; and return '0' otherwise. We label the checks that are done by the ideal functionality in roman numerals.

$\mathcal{F}_{\mathsf{EPID}}$ SETUP: On input $(\text{SETUP}, \mathtt{SID})$ from the issuer $\mathcal{I}$, $\mathcal{F}_{\mathsf{EPID}}$ verifies that $(\mathcal{I}, \mathtt{SID}') = \mathtt{SID}$ and outputs $(\text{SETUP}, \mathtt{SID})$ to $\mathcal{S}$. $\mathcal{F}_{\mathsf{EPID}}$ receives from the simulator $\mathcal{S}$ the algorithms Kgen, sig, ver, $\mathsf{Identify}$and revoke. These algorithms are responsible for generating keys for honest signers, creating signatures for honest signers, verifying the validity of signatures, checking whether a signature was generated by a given key, and updating the revocation lists respectively. $\mathcal{F}_{\mathsf{EPID}}$ stores the algorithms, checks that the algorithms ver, $\mathsf{Identify}$ and Revoke are deterministic [Check I], and outputs $(\text{SETUPDONE}, \mathtt{SID})$ to $\mathcal{I}$.
$\mathcal{F}_{\mathsf{EPID}}$ JOIN:

1. JOIN REQUEST: On input (JOIN, SID, JSID) from a signer Signer, create a join session ⟨JSID, Signer, request⟩. Output (JOINSTART, SID, JSID, Signer) to $\mathcal{S}$.

2. JOIN REQUEST DELIVERY: Proceed upon receiving delivery notification from $\mathcal{S}$ by updating the session record to ⟨JSID, Signer, delivery⟩.

   - If $\mathcal{I}$ or Signer is honest and ⟨Signer, $\star$, $\star$⟩ is already in Member List ML, output $\perp$ [Check II].
   - Output (JOINPROCEED, SID, JSID, Signer) to $\mathcal{I}$.

3. JOIN PROCEED: Upon receiving an approval from $\mathcal{I}$, $\mathcal{F}_{\mathsf{EPID}}$ updates the session record to ⟨JSID, SID, Signer, complete⟩.
   Then it outputs (JOINCOMPLETE, SID, JSID) to $\mathcal{S}$.

4. KEY GENERATION: On input (JOINCOMPLETE, SID, JSID, tsk) from $\mathcal{S}$.

   - If the signer is honest, set $tsk = \perp$, else verify that the provided tsk is eligible by performing the following two checks that are described above:
     - $\mathsf{CheckTtdHonest}(tsk)$=1 [Check III].
     - $\mathsf{CheckTtdCorrupt}(tsk)$=1 [Check IV].
   - Insert ⟨Signer, tsk⟩ into Member List ML, and output JOINED.

$\mathcal{F}_{\mathsf{EPID}}$ SIGN:

1. SIGN REQUEST: On input a request (SIGN, SID, SSID, Signer, msg) from the signer, the ideal functionality calculates the signature identifier sid and aborts if $\mathcal{I}$ is honest and no entry ⟨Signer, $\star$⟩ exists in ML [Check V], else creates a sign session ⟨SSID, Signer, msg, request⟩ and outputs (SIGNSTART,
   SID, SSID, Signer, msg)) to $\mathcal{S}$.

2. SIGN PROCEED: On input (SIGNPROCEED, SID, SSID, msg) from Signer, $\mathcal{F}_{\mathsf{EPID}}$ outputs (SIGNCOMPLETE, SID, SSID, KRL, SRL, msg) to $\mathcal{S}$, where KRL and SRL represent the key and the signature revocation lists respectively.

3. SIGNATURE GENERATION: On input (SIGNCOMPLETE, SID, SSID, Σ, KRL, SRL) from $\mathcal{S}$, if Signer is honest then $\mathcal{F}_{\mathsf{EPID}}$ will:

   - Ignore an adversary's signature Σ, and generate the signature for a fresh or established tsk.
   - Check $\mathsf{CheckTtdHonest}(tsk)$=1 [Check VI], and store ⟨Signer, tsk⟩ in DomainKeys.
   - Generate the signature $\Sigma \leftarrow \mathrm{sig}(tsk, msg)$.

- Check ver$(\Sigma, msg, \texttt{KRL}, \texttt{SRL})$=1 [Check VII], and check $\mathsf{Identify}(\Sigma, msg, tsk) = 1$ [Check VIII].
- Check that there is no signer other than `Signer` with key tsk' registered in `ML` or `DomainKeys` such that $\mathsf{Identify}(\Sigma, msg, tsk')$=1 [Check IX].
- For all $(\Sigma^*, msg^*) \in \texttt{SRL}$, find all $(tsk^*, \texttt{Signer}^*)$ from `ML` and `DomainKeys` such that $\mathsf{Identify}(\Sigma^*, msg^*, tsk^*) = 1$
  - Check that no two distinct keys tsk^* trace back to Σ^* [Check X].
  - Check that no pair $(tsk^*, \texttt{Signer})$ was found [Check IX].
- If `Signer` is honest, then store $\langle \Sigma, \texttt{Signer}, sid \rangle$ in Signed and output (SIGNATURE, `SID`, `SSID`, Σ, msg, `KRL`, `SRL`).

$\mathcal{F}_{\mathsf{EPID}}$ VERIFY: On input (VERIFY, `SID`, msg, Σ, `KRL`, `SRL`), from a party $\mathcal{V}$ to check whether Σ is a valid signature on sid, `KRL` and `SRL`, the ideal functionality does the following:

- Extract all pairs $(tsk', \texttt{Signer}')$ from the `DomainKeys` and `ML`, for which $\mathsf{Identify}(tsk', \Sigma, msg)$=1. Set $b = 0$ if any of the following holds:
  - More than one key tsk_i was found [Check XII].
  - $\mathcal{I}$ is honest and no pair $(tsk, \texttt{Signer})$ was found [Check XIII].
  - An honest `Signer` was found, but no entry $\langle \star, \texttt{Signer}, msg, str \rangle$ was found in Signed [Check XIV].
  - There is a key $tsk^* \in \texttt{KRL}$, such that $\mathsf{Identify}(\Sigma, msg, tsk^*)$=1 and no pair $(tsk, \texttt{Signer})$ for an honest `Signer` was found [Check XV].
  - For some matching tsk and $(\Sigma^*, msg^*) \in \texttt{SRL}$, $\mathsf{Identify}(tsk, \Sigma^*, msg^*) = 1$ [Check XVI]
- If $b \neq 0$, set $b \leftarrow$ver$(\Sigma, msg, \texttt{SRL}, \texttt{KRL})$.
- Add $\langle \Sigma, msg, \texttt{KRL}, \texttt{SRL}, b \rangle$ to VerResults, and output (VERIFIED, `SID`, b) to $\mathcal{V}$.

$\mathcal{F}_{\mathsf{EPID}}$ REVOKE: On input $(tsk^*, \texttt{KRL})$, the ideal functionality replaces `KRL` with $\texttt{KRL} \cup tsk^*$. On input $(\Sigma^*, msg^*, \texttt{SRL})$, the ideal functionality replaces `SRL` with $\texttt{SRL} \cup \Sigma^*$ after verifying Σ^*.

We emphasize that our model catches all the required EPID security properties. The *Correctness* property is achieved since honestly generated signatures always pass through verification check via the algorithm ver$(\Sigma, msg, \texttt{KRL}, \texttt{SRL})$ [Check VIII]. This, in the real world, means that honestly generated signatures are always accepted by the verifier and not being revoked. The *Anonymity* property is guaranteed due to the random choice of the key tsk if the key belongs to an honest signer. In the case of a corrupt signer, the simulator is allowed to provide a signature that may reveal the signer's identity, as the signing key can be extracted from the respective signer key pair. This reflects that the

anonymity of an EPID signer is guaranteed only when the signer is honest. The *Unforgeability* property is due to [Check XIII] and [Check XV] and the *Non-frameability* property is due to the [Check IX]. Furthermore, CheckTtdHonest prevents registering an honest tsk in the Join interface that matches an existing signature so that conflicts can be avoided and signatures can always be traced back to the original signer [Check III]. This ensures that honestly generated signatures are not revoked due to the identified algorithm being deterministic in our model. CheckTtdCorrupt [Check IV] is done when storing a new tsk that belongs to a corrupt Signer, it checks that the new tsk does not break the identifiability of signatures, i.e., it checks that there is no other known Signer key tsk', unequal to tsk, such that both keys are identified as the owner of a signature.

5 UC Security Proof of the EPID Scheme

In this section, we provide a high-level description of the UC-based security proof of our hash-based EPID scheme, which was described in Section 3. We present a sequence of games to show that there exists no environment $\mathcal{E}$ that can distinguish the real-world protocol denoted by Π with an adversary $\mathcal{A}$, from the ideal world $\mathcal{F}$ with a simulator $\mathcal{S}$. Each of these games contains further checks that guarantee a desired security requirement while proving game-to-game indistinguishability. At the end of the proof, we showcase that our real protocol guarantees the desired security requirements that the ideal world provides.

We start with the real-world protocol execution in Game 1. In the next game, we construct one entity C that runs the real-world protocol for all honest parties. Then we split C into two pieces, an ideal functionality $\mathcal{F}$ and a simulator $\mathcal{S}$ that simulates the real-world parties. Initially, we start with an "empty" functionality $\mathcal{F}$. With each game, we gradually change $\mathcal{F}$ and update $\mathcal{S}$ accordingly, moving from the real world to the ideal world, and culminating to the full $\mathcal{F}_{\mathsf{EPID}}$ being realized as part of the ideal world, thus, proving our proposed security model presented in § 4. The endmost goal of our proof is to prove the indistinguishability between Game 1 and Game 14, i.e., between the complete real world and the fully functional ideal world. This is done by proving that each game is indistinguishable from the previous one starting from Game 1 to reach Game 14. As aforementioned, our proof starts with setting up the real-world games (Game 1 and Game 2), followed by introducing the ideal functionality in Game 3. At this stage, the ideal functionality $\mathcal{F}$ only forwards its inputs to the simulator which simulates the real world. From Game 4 onward, $\mathcal{F}$ starts executing the setup interface on behalf of the Issuer. Moving on to Game 5, $\mathcal{F}$ handles simple verification and consistency checks without performing any detailed checks at this stage; i.e., it only checks if the signer belongs to a revocation list. In Games 6-7, $\mathcal{F}$ executes the join interface while performing checks to maintain the registered keys' consistency. It also adds checks that allow only the devices that have successfully been enrolled to create signatures. Game 8

proves the anonymity of EPID by letting $\mathcal{F}$ handle the sign queries on behalf of an honest `Signer` using freshly generated random key instead of running the sign algorithm using the signer's signing key. At the end of this game, we prove that an external environment will notice no change from previous games where the real-world sign algorithm was executed. Now moving to Games 9 - 14, we let $\mathcal{F}$ perform all other checks that are explained in our UC model that ends with the ideal functionality $\mathcal{F}_{\mathsf{EPID}}$ defined in § 4.

Proof. **Game 1 (Real Word)**: This is the real world protocol.

Game 2 (Transition to the Ideal World): An entity C is introduced. C receives all inputs from the honest parties and simulates the real-world protocol for them. This is equivalent to Game 1, as this change is invisible to $\mathcal{E}$.

Game 3 (Transition to the Ideal World with Different Structure): We now split C into two parts, $\mathcal{F}$ and $\mathcal{S}$, where $\mathcal{F}$ behaves as an ideal functionality. It receives all the inputs and forwards them to $\mathcal{S}$, which simulates the real-world protocol for honest parties and sends the outputs to $\mathcal{F}$. $\mathcal{F}$ then forwards these outputs to $\mathcal{E}$. This game is essentially equivalent to Game 2 with a different structure which is invisible to $\mathcal{E}$.

Game 4 ($\mathcal{F}$ handles the setup): $\mathcal{F}$ now behaves differently in the setup interface, as it stores the algorithms defined in § 4 that are provided by $\mathcal{S}$ from real-world protocol. $\mathcal{F}$ stores the algorithms, and checks that the algorithms ver, $\mathsf{Identify}$ and Revoke are deterministic [Check I], such check is indistinguishable from the real world since the algorithms are adopted from the real-world protocol. $\mathcal{F}$ also performs checks and ensures that the structure of `SID`, which represents the issuer's unique session identifier defined in § 4, is correct for an honest $\mathcal{I}$, and aborts if not. When $\mathcal{I}$ is honest, $\mathcal{S}$ will start simulating it. Since $\mathcal{S}$ is now running the Issuer, it knows its secret key. In case $\mathcal{I}$ is corrupt, $\mathcal{S}$ extracts $\mathcal{I}$'s secret key from $\pi_{\mathcal{I}}$ when the issuer registers his key with $\mathcal{F}_{ca}$, a common certificate authority functionality that is available to all parties and controlled by the simulator, then proceeds to the setup interface on behalf of $\mathcal{I}$. By the simulation soundness of $\pi_{\mathcal{I}}$, this game transition is indistinguishable from the previous game (Game 4 $\approx$ Game 3).

Game 5 ($\mathcal{F}$ handles the verification and the revocation checks): $\mathcal{F}$ now performs the verification and the revocation checks instead of forwarding them to $\mathcal{S}$. There are no protocol messages and the outputs are exactly as in the real-world protocol. Knowing the lists `KRL` and `SRL` for corrupt signers, $\mathcal{F}$ can perform the revocation checks, and the outcomes of these checks are equal to the real-world protocol (Game 5 $\approx$ Game 4).

Game 6: ($\mathcal{F}$ handles the join queries): $\mathcal{F}$ stores in its records the members that have joined. If $\mathcal{I}$ is honest, $\mathcal{F}$ stores the secret key tsk, extracted from $\mathcal{S}$, for a corrupt signer. $\mathcal{S}$ always has enough information to simulate the real-world protocol except when the issuer is the only honest party. In this case, $\mathcal{S}$ does not know who initiated the join, and so cannot make a join query with $\mathcal{F}$ on the signer's behalf. Thus, to deal with this case, $\mathcal{F}$ can safely choose any corrupt signer and put it into `ML`. The identities of signers are only used for creating signatures for honest signers, so corrupted signers do not matter. In

the case that the signer is already registered in $\texttt{ML}$, $\mathcal{F}$ would abort the protocol [Check II], but $\mathcal{I}$ will have already tested this case before continuing with the proceeding with the join query. Hence $\mathcal{F}$ will not abort. Thus in all cases, $\mathcal{F}$ and $\mathcal{S}$ can interact to simulate the real-world protocol, so Game 6 $\approx$ Game 5.

Game 7: ($\mathcal{F}$ performs pre-sign checks): If $\mathcal{I}$ is honest, then $\mathcal{F}$ now only allows members that joined to sign [Check V]. An honest signer will always check whether it has successfully joined (i.e., has been issued a valid credential) before signing in the real-world protocol, so there is no difference for honest signers. Therefore Game 7 $\approx$ Game 6.

Game 8: ($\mathcal{F}$ handles the sign queries by simulating the `Signer` without knowing the secret): We now transform $\mathcal{F}$ such that it internally handles the signing queries of honest signers instead of merely forwarding them to $\mathcal{S}$ that simulates the `Signer` and creates a signature using the `Signer`'s key $tsk = sk_u$ as in the games before. When the `Signer` is honest, $\mathcal{F}$ creates the signatures internally as follows: It chooses a new key $key \leftarrow \{0,1\}^n$ per signature and then runs the sign algorithm $\text{sig}(key, msg)$ defined in Section 4 for that fresh key. To prove the anonymity of our EPID scheme, we show that if there exists an environment that can distinguish a signature of an honest party using $tsk = sk_u$ from a signature using a random key $key \leftarrow \{0,1\}^n$, then the environment can break the pseudorandom property of the function F.

Suppose that $\mathcal{E}$ is given tuples $\Sigma = (str,\ sst,\ com,\ r,\ \forall_{j\in[1,J]}\ A_j,\ \pi_{\texttt{E}})$, where J denote the total number of the revoked signatures. In the reduction, we have to be able to simulate the `Signer` u without knowing the secret sk_u. Let key be a randomly sampled key from $\{0,1\}^n$ that will be used to generate signatures on behalf of the honest `Signer` rather than using the real `Signer` secret key sk_u. Since the issuer's secret key msk can be extracted from the issuer's zero-knowledge proof $\pi_{\mathcal{I}}$ for the correctness of the master secret and public key pair due to the soundness of the proof $\pi_{\mathcal{I}}$ and getting access to $\mathcal{F}_{crs}$, a common reference string functionality that provides participants with all system parameters. Then a credential can be created on $et'_u = F(key,\ gid)$ by running the signing algorithm of F-SPHINCS+ with the input (et'_u, msk, gp). After getting a credential on et'_u, sst will be calculated as functions of key such that $sst = F(key,\ sid)$ where $sid = H(msg, str)$. All other parts of the signature follow the same real-world protocol (i.e. when using the `Signer`'s sk_u). The commitment com is calculated as our defined sign algorithm and the proof $\pi_{\texttt{E}}$ can then be perfectly simulated using the random secret key. Due to the zero-knowledge property of the proof $\pi_{\texttt{E}}$ and the pseudorandom outputs of the function F, we argue that an external environment cannot distinguish between 1) a signature generated using the `Signer`'s (sk_u, et_u) and 2) a signature generated by a random (key, et'_u). The signature will not match one of the revoked signatures $\Sigma_j \in \texttt{SRL}$ with an overwhelming probability, because the probability that Σ matches one of the revoked signatures is that key matches one of the keys that were used to generate Σ_j which is $J/2^n$. For large n, this becomes negligible. With a high probability the signature that is generated from a random key r will not be revoked, so Game 8 $\approx$ Game 7.

Game 9: ($\mathcal{F}$ performs pre-signing checks): When storing a new $tsk =$

sk_u, $\mathcal{F}$ checks $\mathsf{CheckTtdCorrupt}(tsk)$=1 [Check IV] or $\mathsf{CheckTtdHonest}(tsk)$=1 [Check III] . We want to show that these checks will always pass. A valid signatures always satisfy $et_u = F(sk_u, gid)$, $sst = F(sk_u, sid)$ and $(gr_u, \mathbf{S}) \leftarrow \mathsf{F\text{-}SPHINCS+.sign}(et_u, msk, gp)$ where et_u corresponds to a signing key sk_u. In the real-world protocol, we have $tsk = sk_u$. By the soundness property of $\pi_{\mathtt{E}}$ that is proved in § 3.4, there exists one sst that matches this signature. Thus, $\mathsf{CheckTtdCorrupt}(tsk) = 1$ will always give the correct output. Also, due to the large min-entropy of the uniform distribution the probability that sampling a selected sk_u is negligible for large n with probability equal to $1/2^n$, with overwhelming probability, there doesn't exist a signature already using the same sk_u, which implies that $\mathsf{CheckTtdHonest}(tsk) = 1$ will give the correct output with overwhelming probability. Hence, Game 9 $\approx$ Game 8.

Game 10: ($\mathcal{F}$ checks the correctness of the protocol): In this game $\mathcal{F}$ checks that any honestly generated signature $\Sigma = (str, sst, com, \pi_{\mathtt{E}})$ is always valid due to the completeness property of π_E and the correctness of the F-SPHINCS+ signature. A valid proof π_E on the credential ensures that the credential has the correct structure and always leads to the correct extraction of the issuer public key rpk due to the soundness of $\pi_{\mathtt{E}}$ and the correctness of the F-SPHINCS+ signature. Second, $\mathcal{F}$ makes sure $\mathsf{identify}(tsk, \Sigma, msg) = 1$, which is also achieved in the real-world protocol due to the soundness of $\pi_{\mathtt{E}}$. $\mathcal{F}$ checks, using its internal records `ML` and `DomainKeys` that honest users are not sharing the same secret key tsk. The soundness of $\pi_{\mathtt{E}}$ also ensures that the honestly generated signatures will not match any revoked signature, this is checked by $\mathcal{F}$ [Check XV] and [Check XVI] in the ideal world. This due to the calculations of $A_j = F(F(sk_u, sid_j), r)$ and $B_j = F(sst_j, r)$ for $\forall\ \Sigma_j \in \mathtt{SRL}$, where $sst_j = F(sk_j, sid_j)$ and $r \xleftarrow{R} \{0,1\}^n$, if $A_j \neq B_j$ then $F(sk_j, sid_j) \neq F(sk_u, sid_j)$ and hence $sk_j \neq sk_u$ due to the collision resistance property of the function F. Therefore Game 10 $\approx$ Game 9.

Game 11 ($\mathcal{F}$ checks that valid signatures are deterministic): Add [Check IX] to ensure that there are no multiple sk_u values matching to one signature. However, since there exists only one sk_u such that $sst = F(sk_u,\ sid) \wedge et_u = F(sk_u,\ gid) \wedge com = H_1(s||pk_h||\cdots||rpk)$ due to collision resistance of the function F and the binding property of the commitment com, thus two different signatures cannot share the same sk_u. In general, if there exist two signatures sharing the same sk_u then this breaks the soundness of $\pi_{\mathtt{E}}$, thus any valid signature should be identified to one sk_u. Thus Game 11 $\approx$ Game 10.

Game 12 ($\mathcal{F}$ checks the unforgeability of a credential): To prevent accepting signatures whose corresponding credentials are not issued by the honest issuer, $\mathcal{F}$ adds a further check [Check XIII]. This follows the unforgeability property of the F-SPHINCS+ scheme as analysed in Appendix A. In our EPID scheme, a membership credential is the issuer's F-SPHINCS+ signature on the `Signer`'s public key, so we get Game 12 $\approx$ Game 11.

Game 13 ($\mathcal{F}$ checks the unforgeability of EPID signatures): [Check XIV] is added to $\mathcal{F}$ to prevent anyone forging a signature, which should be signed under honest signer's group signing key $gsk_u = (sk_u,\ gr_u,\ \mathbf{S})$. If such

a signature is verified then due to the binding property of the commitment scheme used to generate $com = H_1(s||pk_h||\cdots||rpk)$, where s is a nonce and $pk_h, \cdots, rpk$ are the root values of a chained M-FORS trees. These keys are used to verify the F-SPHINCS+ signature $\mathbf{S}$. The signature should correctly verify on an entry token $(et_u||gr_u)$ under the first committed verification key pk_h, where $et_u = F(sk_u,\ gid)$ for some honest signer's signing key sk_u. We argue that any modification in the committed verification keys would end with a failed verification of the signature $\mathbf{S}$ on the user entry token $(et_u||gr_u)$ due to the correctness of F-SPHINCS+. Now suppose that an adversary knows the credential $(gr_u, \mathbf{S})$ but not sk_u. The adversary chooses a random sk'_u and proceeds with the MPCitH NIZKPs for the construction of $et'_u = F(sk'_u, gid)$. Due to the soundness of $\pi_{\mathtt{E}}$, the proof cannot be simulated unless $(et'_u||gr_u)$ verifies to be correctly signed under $pk_h, \cdots, rpk$ used in com. This would only happen if $et'_u = et_u$ due to F-SPHINCS+ being a q-EU-CMA secure signature (see Appendix A for a proof). Therefore sk'_u should match sk_u. The probability of this to happen is $1/2^n$, then the advantage of the adversary is negligible for large n. Therefore, Game 13 $\approx$ Game 12 with overwhelming probability.

Game 14 ($\mathcal{F}$ checks the correct revocation): [Check XV] and [Check XVI] are added to $\mathcal{F}$ to ensure that honestly generated signatures are not being revoked. If there exists a matching revoked key $sk^*_u \in \mathtt{KRL}$ which belongs to an honest signer and is not revoked, then this breaks the collision resistance property of F. Suppose that there exists a $\Sigma_j \in \mathtt{SRL}$, such that $\text{Identify}(sk^*_u, \Sigma_j, msg_j) = 1$, i.e., the equation $A_j = B_j$ holds. This means $F(F(sk^*_u, sid_j), r) = F(sst_j, r)$, where $sst_j = F(sk_j, sid_j) \in \Sigma_j$. Due to the soundness of the MPCitH proof of F, sk^*_u will match to sk_j which was used to generate $\Sigma_j \in \mathtt{SRL}$. However, as we assumed that sk^*_u is not revoked, so $A_j \neq B_j$ will hold for all $\Sigma_j \in \mathtt{SRL}$. Therefore, Game 14 $\approx$ Game 13. □

6 Implementation and Comparison

We implemented the signature revocation algorithm to show its feasibility and assess its performance[1]. The implementation was in C++ and used much of the code provided by Chen *et al.* [22] in support of their paper on hash-based group signatures. This code was itself based on the Picnic KKW scheme (namely `picnic3`) and used some subroutines from the Picnic implementation [48] submitted to the NIST Post-Quantum Cryptography Standardization project [44]. In their implementation, Chen *et al.* used two different security parameters, i.e. $n = 129$ and $n = 255$ and we use these same values for our tests. For the MPC-in-the-Head parameters, we use those from the Picnic implementations. Namely for $n = 129$, $NR = 250$ and $NO = 36$, while for $n = 255$, $NR = 601$ and $NO = 68$. NR and NO are the total number of MPC instances and the number of opened MPC instances respectively.

We measure the performance of our EPID revocation scheme based on our C++ implementation. The programs were compiled using the GNU GCC com-

[1] The revocation code is available at: `https://github.com/UoS-SCCS/HB_EPID_Revocation`.

piler [32] version 12.3.0 and executed on a laptop (Intel i7-8850H CPU @2.6GHz : 32Gb RAM) with the Ubuntu operating system. Although the CPU has multiple cores, the timings were obtained using a single core. The performance figures are given in Table 1 and are the averages for 10 runs. The timings were obtained using the C++ timing routines. It should be noted that, unlike the credential signing times these times are independent of the group size. They are, as to be expected, approximately linear in the revocation list size.

Table 1: Revocation test results for different signature revocation list sizes (times are in seconds and sizes in MB).

n	SRL size	sign	verify	$\pi_{\mathtt{R}}$ size
129	10	0.7	0.3	0.12
	100	6.7	3.2	1.0
	500	34.3	16.3	5.2
	1000	67.5	32.5	10.4
255	10	3.4	1.6	0.43
	100	33.1	15.7	3.9
	500	164.2	78.0	19.2
	1000	328.0	155.8	38.2

Table 2: Test results from Chen *et al.* [22] for various group sizes (times are in seconds and sizes in MB).

n	group size	sign	verify	sig. size
129	2^{10}	8.7	4.4	0.56
	2^{20}	12.9	6.3	0.83
	2^{40}	21.4	10.2	1.39
	2^{60}	29.4	13.9	1.95
255	2^{10}	36.4	17.3	2.28
	2^{20}	53.0	25.6	3.40
	2^{40}	89.4	43.0	5.65
	2^{60}	125.1	60.0	7.91

The timings are consistent with those obtained by Chen *et al.* but are very slow when compared against the results reported for the reference Picnic

implementation. We are currently investigating this discrepancy. As, to support confidentiality, the code allows the inputs and outputs to the LowMC function to be masked this introduces extra overheads when calculating each single LowMC function and this may partially explain the discrepancy. This idea is supported by the results of profiling the code when approximately 60% of the time is reportedly spent in the (Picnic) tape generation and bit manipulation routines.

To enable comparison with other post-quantum EPID schemes we combine the measurements from Chen *et al.* [22] (reproduced in Table 2) with those that we obtained for $\pi_{\mathtt{R}}$ to give us an estimate of the signing times and signature sizes for our EPID scheme. In Table 3 we show a comparison with several lattice-based EPID implementations for which data is available. For these implementations figures were given for a signature revocation list containing 1000 items and to make the comparison we use our figures for 1000 items and group sizes of 2^{40} and 2^{60}. While our scheme is clearly better when considering the user private key and signature sizes, the situation is less clear for the timings; We are certainly on a par with the lattice-based schemes for $n = 129$, but lose out when the security parameter is increased to $n = 255$. However, it should be noted that this is a reference implementation in C++ with no attempt at optimisation or adaptation to use more efficient, but processor specific, processor instructions. There is much scope for improving these timings.

Table 3: Comparison with lattice-based EPID schemes (signature sizes in MB and times in seconds).

scheme	private-key	signing	verification	sig. size
LEPID [20]	58 KB	200	80	$9^{\dagger}$
LEPID [30]	100 KB	374	121	854
$n = 129, GS = 2^{40}$	17 bytes	89	43	11.8
$n = 129, GS = 2^{60}$	17 bytes	97	46	12.4
$n = 255, GS = 2^{40}$	32 bytes	417	199	43.9
$n = 255, GS = 2^{60}$	32 bytes	453	216	46.1

† This figure is estimated from the signature revocation size in [30] and the signature size in [23].

7 Conclusions

This paper proposes a new EPID scheme from symmetric primitives. Following recent research on group signatures and direct anonymous attestation, we make use of a modified SPHINCS+ signature as a group membership credential and use an MPCitH-based signature to prove the possession of that credential. This choice allows our EPID scheme to handle a large group size (up to 2^{60}), which is suitable for rapidly increasing cyber security and trusted computing applications. Our EPID scheme has an efficient signature-based revocation method. The security of the EPID scheme is proved under the UC framework. We present several optimizations to improve performance and provide a prototype imple-

mentation. For future work, we will investigate ways of improving performance and obtaining more practical benchmarks.

Acknowledgments

We thank the European Union's Horizon research and innovation program for support under grant agreement numbers: 779391 (FutureTPM), 952697 (ASSURED), 101019645 (SECANT), 101069688 (CONNECT), 101070627 (REWIRE) and 101095634 (ENTRUST). These projects are funded by the UK government's Horizon Europe guarantee and administered by UKRI. We also thank the National Natural Science Foundation of China for support under grant agreement numbers: 62072132 and 62261160651. Finally, we appreciate the valuable comments from the anonymous reviewers of PQCrypto 2024; particularly, the suggestion that removing B_j values from a signature would improve the performance of our scheme.

References

[1] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-Head. In *CRYPTO*, pages 581–615, 2023.

[2] Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In *Public-Key Cryptography - PKC*, pages 266–297, 2021.

[3] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *EUROCRYPT*, pages 103–128, 2019.

[4] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In *ACM CCS*, pages 2129–2146, 2019.

[5] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In *ACM CCS*, pages 2025–2038, 2020.

[6] Chinmoy Biswas, Ratna Dutta, and Sumanta Sarkar. An efficient post-quantum secure dynamic EPID signature scheme using lattices. *Multimedia Tools and Applications*, pages 1–30, 2023.

[7] Bruno Blanchet et al. Modeling and verifying security protocols with the applied Pi calculus and ProVerif. *Foundations and Trends® in Privacy and Security*, 1(1-2):1–135, 2016.

[8] Dan Boneh, Saba Eskandarian, and Ben Fisch. Post-quantum EPID signatures from symmetric primitives. In *CT-RSA*, pages 251–271, 2019.

[9] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *ACM CCS*, pages 168–177, 2004.

[10] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Alessandro Sorniotti. A framework for practical anonymous credentials from lattices. Cryptology ePrint Archive, Paper 2023/560, 2023. `https://eprint.iacr.org/2023/560`.

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

[12] Ernie Brickell and Jiangtao Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 21–30, 2007.

[13] Ernie Brickell and Jiangtao Li. Enhanced Privacy ID from bilinear pairing. Cryptology ePrint Archive, Paper 2009/095, 2009. https://eprint.iacr.org/2009/095.

[14] Ernie Brickell and Jiangtao Li. Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. *International Journal of Information Privacy, Security and Integrity 2*, 1(1):3–33, 2011.

[15] Ernie Brickell and Jiangtao Li. Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities. *IEEE Trans. Dependable Secur. Comput.*, 9(3):345–360, 2012.

[16] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. In *IEEE Symposium on Security and Privacy*, pages 901–920. IEEE, 2017.

[17] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In *Public-Key Cryptography – PKC*, pages 234–264. Springer, 2016.

[18] 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 *ACM CCS*, pages 1825–1842, 2017.

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

[20] Liquan Chen, Zeyu Xu, Tianyang Tu, and Zhongmin Wang. Lattice-based privacy enhanced identity protocol for SDO services. In *International Conference on Signal and Image Processing (ICSIP)*, pages 609–613, 2023.

[21] Liqun Chen, Changyu Dong, Nada El Kassem, Christopher JP Newton, and Yalan Wang. Hash-based direct anonymous attestation. In *PQCrypto*, pages 565–600, 2023.

[22] Liqun Chen, Changyu Dong, Christopher JP Newton, and Yalan Wang. Sphinx-in-the-head: Group signatures from symmetric primitives. *ACM Transactions on Privacy and Security*, 2023.

[23] Liqun Chen, Nada El Kassem, Anja Lehmann, and Vadim Lyubashevsky. A framework for efficient lattice-based DAA. In *Proceedings of the 1st ACM Workshop on Workshop on Cyber-Security Arms Race*, pages 23–34, 2019.

[24] Ming-Shing Chen, Yu-Shian Chen, Chen-Mou Cheng, Shiuan Fu, Wei-Chih Hong, Jen-Hsuan Hsiang, Sheng-Te Hu, Po-Chun Kuo, Wei-Bin Lee, Feng-Hao Liu, and Justin Thaler. Preon: zk-SNARK based signature scheme. NIST PQ Signatures submissions, 2023. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/Preon-spec-web.pdf.

[25] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. Cachequote: Efficiently recovering long-term secrets of SGX EPID via cache attacks. *ICAR Transactions on Cryptographic Hardware and Embedded Systems*, pages 171–191, 2018.

[26] Cyprien Delpech de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. BBQ: using AES in Picnic signatures. In *Selected Areas in Cryptography - SAC*, pages 669–692, 2019.

[27] Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge MPCitH-based arguments. In *ACM CCS*, pages 3022–3036, 2021.

[28] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. Shorter signatures based on tailor-made minimalist symmetric-key crypto. In *ACM CCS*, pages 843–857, 2022.

[29] Nada El Kassem. *Lattice-based direct anonymous attestation.* PhD thesis, University of Surrey, 2020.

[30] Nada El Kassem, Luís Fiolhais, Paulo Martins, Liqun Chen, and Leonel Sousa. A lattice-based enhanced privacy ID. In *Information Security Theory and Practice, WISTP 2019*, pages 15–31. Springer, 2020.

[31] Antonio Faonio, Dario Fiore, Luca Nizzardo, and Claudio Soriente. Subversion-resilient enhanced privacy ID. In *CT-RSA*, pages 562–588. Springer, 2022.

[32] Free Software Foundation, Inc. GCC, the GNU Compiler Collection. `https://ggcc.gnu.org`, 2022.

[33] Shihui Fu and Guang Gang. Polaris: Transparent succinct zero-knowledge arguments for R1CS with efficient verifier. In *Proceedings on Privacy Enhancing Technologies*, pages 544–564, 2022.

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

[35] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.

[36] ISO/IEC 10118-2:2010. Information technology — Security techniques — Hash-functions — Part 2: Hash-functions using an n-bit block cipher. Standard, International Organization for Standardization, Geneva, CH, 2010.

[37] ISO/IEC 20008-2:2013. Information technology — Security techniques — Anonymous digital signatures — Part 2: Mechanisms using a group public key. Standard, International Organization for Standardization, Geneva, CH, 2013.

[38] Daniel Kales and Greg Zaverucha. Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures. Cryptology ePrint Archive, Paper 2022/588, 2022. https://eprint.iacr.org/2022/588.

[39] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS*, pages 525–537, 2018.

[40] Seongkwang Kim, Jincheol Ha, Mincheol Son, Byeonghak Lee, Dukjae Moon, Joohee Lee, Sangyub Lee, Jihoon Kwon, Jihoon Cho, Hyojin Yoon, et al. AIM: symmetric primitive for shorter signatures with stronger security. In *ACM CCS*, pages 401–415, 2023.

[41] Leslie Lamport. Constructing digital signatures from a one-way function. *Tech. Report: SRI International Computer Science Laboratory*, 1979.

[42] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. Lattice-based zero-knowledge proofs and applications: shorter, simpler, and more general. In *CRYPTO*, pages 71–101. Springer, 2022.

[43] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO*, pages 218–238, 1989.

[44] NIST. Post-quantum cryptography standardization. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization, 2017-2022.

[45] NIST. NIST announces first four quantum-resistant cryptographic algorithms. https://nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms, 2022.

[46] Muhammad Usama Sardar, Christof Fetzer, et al. Towards formalization of enhanced privacy ID (EPID)-based remote attestation in Intel SGX. In *Euromicro Conference on Digital System Design (DSD)*, pages 604–607, 2020.

[47] TCG. TPM 2.0 library specification. https://trustedcomputinggroup.org/resource/tpm-library-specification/.

[48] Zaverucha, Ramacher, Kales, and Goldfeder. Reference implementation of the Picnic post-quantum signature scheme. https://github.com/Microsoft/Picnic, 2020.

[49] Gerg Zaverucha. The Picnic signature algorithm specification. Supporting Documentation in https://github.com/Microsoft/Picnic, 2020.

Experiment $\text{Exp}^{q\text{-EU-CMA}}_{\text{SIG},A}(n)$

- $(sk, pk) \leftarrow kg$
- $(M^*, \sigma^*) \leftarrow A^{sign(sk,\cdot)}(pk)$, and A can query the $sign$ Oracle at most q times.
- Return 1 iff $vf(pk, M^*, \sigma^*) = 1 \wedge M^* \notin \{M_i\}_{i=1}^{q}$

Figure 4: q-EU-CMA game.

Appendix

A Security Analysis: F-SPHINCS+

The standard security definition for digital signature schemes is existential unforgeability under adaptive chosen-message attacks (EU-CMA). It can be extended to a few-time signature by limiting the adversary's call to the sign oracle to q times where q is the maximum number of signatures that the few-time signature scheme is allowed to generate for each signing key. Let $SIG = (kg, sign, vf)$ be a q-time signature scheme, Figure 4 shows the q-EU-CMA game.

Definition 1 (q-EU-CMA)**.** *Let SIG be a digital signature scheme. It is said to be q-EU-CMA secure, if for any adversary A, the following holds:*

$$Succ^{q\text{-}EU\text{-}CMA}_{SIG}(A(n)) = Pr\left[\text{Exp}^{q\text{-}EU\text{-}CMA}_{SIG,A}(n) = 1\right] \leq negl(n)$$

Theorem 1. *For suitable parameters, n, d, k, h, q, the F-SPHINCS+ signature is q^h-EU-CMA secure if:*

- *H_1 is SM-TCR and SM-DSPR secure;*
- *H_2 is TSR secure with at most q queries;*
- *H_3 is ITSR secure with at most q^h queries;*
- *prf is a secure pseudorandom function.*

Proof. To successfully forge a group issuer's signature on a message M chosen by the adversary, there are the following mutually exclusive cases:

1 Let $MD||idx = H_3(M||gr)$ for some gr. In the forged signature, All secret strings corresponding to $MD = p_0||\cdots||p_{k-1}$, i.e. $\{\mathbf{x}^{(i)}_{p_i}\}_{i=0}^{k-1}$, are the same as generated from $\texttt{leaf}_{idx}$'s secret key. This case consists of the following sub-cases:

1.1 The adversary learns all secret strings from signatures obtained in the query phase.

1.2 Some secret strings are not leaked from previous signatures, and for each of them, the adversary either:

1.2.1 learns it by breaking the pseudorandom function that is used to expand the secret key into $\mathbf{x}_i$;

1.2.2 or learns it by looking at their H_1 hash values and find the pre-images.

2 Let $MD||idx = H_3(M||gr)$ for some gr. In the forged signature, some secret strings corresponding to $MD = p_0||\cdots||p_{k-1}$, i.e. $\{\mathbf{x}_{p_i}^{(i)}\}_{i=0}^{k-1}$, are NOT the same as generated from $\texttt{leaf}_{idx}$'s secret key. Then let $\mathbf{S}$ be the list of $h+1$ M-FORS signatures in the forged signature, we can find i such that when verifying the i-th signature ($0 \leq i \leq h$), we obtain the same public key as would be generated by the signer, but for all $0 \leq j < i$, we obtain a different public key as would be generated by the signer. This means:

2.1 The adversary has found at least one second-preimages of H_1 so that some Merkle trees in the ith signature are computed with the second-preimages. They end up having the same roots as the trees computed by the group issuer.

2.2 The adversary knows all secret strings corresponding to the public key produced from verifying the $(i-1)$th signature. This public key is different from the public key at the same location generated by the group issuer. This can be done by either:

2.2.1 learning all from previous signature queries;

2.2.2 or breaking the pseudorandom function;

2.2.3 or finding some pre-images of H_1.

Given the above, we analyze the F-SPHINCS+ signature scheme through a series of games:

Game 0: The original EU-CMA game in which the adversary needs to forge a valid group issuer's signature after q_s queries.

Game 1: Exactly as Game 0 except all output of prf are replaced by truly random n-bit strings. We eliminate from the above list Case 1.2.1 and 2.2.2 by this modification. Since each call to prf uses a secret key and a distinct value as input, assuming prf is a pseudorandom function, we have:

$$|\text{Succ}^{Game0}(A(n)) - \text{Succ}^{Game1}(A(n))| \leq negl(n)$$

Game 2: Game 2 differs from Game 1 in that we consider the adversary lost if the adversary outputs a forgery by breaking the ITSR security of H_3. This modification eliminates from the above list Case 1.1. The winning condition in Figure 4 is changed to:

- Return 1 iff $ITSR(H_3, M^*) = 0 \wedge vf(pk, M^*, \sigma^*) = 1 \wedge M^* \notin \{M_i\}_{i=1}^{q^h}$.

The predicate $ITSR$ is defined as the following:

- Let M^* be the message that the adversary chooses to generate the forgery on, and gr^* the random string used by the adversary to compute $MD^*||idx^* = H_3(M^*||gr^*)$.
- Parse $MD^* = p_0^*||\cdots||p_{k-1}^*$ where each $p_j^* \in [0, 2^d - 1]$. From the above we obtain a set $C^* = ((idx^*, 0, p_0^*), \cdots, (idx^*, k-1, p_{k-1}^*))$.
- For each message queried in the query phase M_i $(1 \leq i \leq q^h)$, and gr_i the random string, compute $MD_i||idx_i = H_3(M_i||gr_i)$ and obtain $C_i = ((idx_i, 0, p_{i,0}), \cdots, (idx_i, k-1, p_{i,k-1}))$.
- Return 1 iff $C^* \subseteq \bigcup_{i=1}^{q^h} C_i$.

We can see that $ITSR(H_3, M^*) = 0$ iff the adversary can break the ITSR security of H_3. Hence, we have:

$$|\text{Succ}^{Game1}(A(n)) - \text{Succ}^{Game2}(A(n))| \leq \text{Succ}^{ITSR}_{H_3,q^h}(A) \leq negl(n)$$

Game 3: Game 3 differs from Game 2 in that we consider the adversary lost if the forgery contains a second preimage for an input to H_1 that was part of a signature returned as a signing-query response. Here the second preimage can be included explicitly in the signature, or implicitly observed when verifying the signature. This eliminates from the above list Case 2.1. Then we have:

$$|\text{Succ}^{Game2}(A(n)) - \text{Succ}^{Game3}(A(n))| \leq \text{Succ}^{SM-TCR}_{H_1,q}(A) \leq negl(n)$$

Game 4: Game 4 differs from Game 3 in that we consider the adversary lost if the adversary outputs a forgery by breaking the TSR security of H_2, which allows the adversary to forge an intermediate signature in $\mathbf{S}$, and then any signature earlier in the chain. This eliminates from the above list Case 2.2.1. The winning condition in Figure 4 is changed to:

- Return 1 iff $TSR(H_2, M^*) = 0 \wedge ITSR(H_3, M^*) = 0 \wedge vf(pk, M^*, \sigma^*) = 1 \wedge M^* \notin \{M_i\}_{i=1}^{q^h}$.

The predicate TSR is defined as the following:

- The adversary chooses an intermediate node in the hyper-tree at address (a, b), and two n-bit string L^*, R^*.
- For each signature obtained in the query phase, if $\mathbf{S}_i$ includes a signature generated using the secret key in node (a, b) over the public key in one of its child node, parse this public key into k blocks, each of d-bit $pk_i = p_{i,0}||\cdots||p_{i,k-1}$, and generate a set $C_i = \{(j, p_{i,j})\}_{j=0}^{k-1}$.
- Compute $pk^* = H_2(\texttt{aux}||k||0||0||L^*||R^*)$, parse pk^* into $p_0^*||\cdots||p_{k-1}^*$, and generate a set $C^* = \{(j, p_j^*)\}_{j=0}^{k-1}$.
- Return 1 iff $C^* \subseteq \bigcup_{i=1}^{q} C_i$.

Note that each M-FORS public key is the root of a Merkle tree generated from pseudorandom strings. Also for each intermediate node in a hyper-tree, it has at most q children, hence no more than q signatures signed by the secret key in this intermediate node can be obtained by the adversary. So $TSR(H_2, M^*) = 0$ iff the adversary can break the TSR security of H_2. Hence, we have:

$$|\mathrm{Succ}^{Game3}(A(n)) - \mathrm{Succ}^{Game4}(A(n))| \leq \mathrm{Succ}^{TSR}_{H_2,q}(A) \leq negl(n)$$

Now the cases in which the adversary can forge a signature are all eliminated except Case 1.2.2 and 2.2.3, which requires the adversary to find a pre-image of at least one hash values produced by H_1. The success probability of finding a pre-image is as analyzed in [4]:

$$\begin{aligned}\mathrm{Succ}^{Game4}(A) &\leq 3 \cdot \mathrm{Succ}^{SM-TCR}_{H_1,p}(A) + \mathrm{Adv}^{SM-DSPR}_{H_1,p}(A) \\ &\leq negl(n)\end{aligned}$$

So overall, the advantage of the adversary is negligible. □

TSR security of H_2

In any case, q signatures can be generated under the secret key of a non-leaf node in the hyper-tree. Assuming the adversary knows all of them, then for each block of the chosen pk^*, the probability of the secret string has been leaked is $1 - (1 - \frac{1}{2^d})^q$, so all secret string have been leaked is $(1 - (1 - \frac{1}{2^d})^q)^k$. For $d = 16, q = 1024, k = 68$, this probability is $2^{-468.87}$, if $k = 35$, this probability is $2^{-210.39}$.

ITSR security of H_3

For a leaf node of the hyper-tree, it may have been used to sign γ signatures out of the total q_s signature queries. So the probability that all secret string of a chosen message M being leaked through query is:

$$\sum_{\gamma}(1 - (1 - \frac{1}{2^d})^{\gamma})^k \binom{q_s}{\gamma} (1 - \frac{1}{q^h})^{q_s - \gamma} \frac{1}{q^{h\gamma}}$$

For $d = 16, q = 1024, k = 68, h = 6, q_s = 2^{60}$, this probability is $2^{-407.32}$, if $k = 35$, this probability is $2^{-208.95}$.