MPC in the head for isomorphisms and group actions

Antoine Joux[0000−0003−2682−6508]

CISPA – Helmholtz Center for Information Security, Saarbrücken, Germany.
joux@cispa.de

Abstract. In this paper, we take inspiration from an invited talk presented at CBCrypto'23 to design identification protocols and signature schemes from group actions using the MPC-in-the-head paradigm. We prove the security of the given identification schemes and rely on the Fiat-Shamir transformation to turn them into signatures.
We also establish a parallel with the technique used for the MPC-in-the-head approach and the seed tree method that has been recently used in some signature and ring signatures algorithms based on group action problems.

1 Introduction

The multiparty computation (MPC) in the head paradigm for zero-knowledge (ZK) protocols was initially introduced in [35] as a tool to provide better theoretical and asymptotic constructions of such ZK protocols. The general setting is the following: given any NP-relation $\mathcal{R}(x, w)$, we want to design a ZK-protocol where a prover $\mathcal{P}$ convinces a verifier $\mathcal{V}$ that he knows a valid witness w for a public value x without revealing any information on w. This (two-party) protocol is constructed from a general MPC protocol where n parties $\mathcal{Z}_1$, $\mathcal{Z}_2$, …, $\mathcal{Z}_n$ check that they collectively share a valid witness w for x. The most frequent approach is to assume that w is encoded in the form $w = w_1 \oplus w_2 \oplus \cdots \oplus w_n$, where each party $\mathcal{Z}_i$ holds the corresponding share w_i, and that there exists an efficient protocol Π to verify that the sharing is correct. In this situation, the MPC-in-the-head prover $\mathcal{P}$ creates a fresh sharing of his secret w, emulates an execution of Π, and commits to the parties' views in this execution. After that, the verifier $\mathcal{V}$ asks for the opening of a subset of these views and verifies that all the opened views are accepting and consistent. If the original multiparty protocol is private against the opened subset, i.e., guarantees that these players cannot conspire to recover the secret, the resulting two-party protocol becomes zero-knowledge.

Other types of sharing techniques can be used instead of the simple $w = w_1 \oplus w_2 \oplus \cdots \oplus w_n$; this is, for example, the case in [28]. Another amusing example of using a different sharing was given during an invited talk at CBCrypto'23, showing the application of MPC-in-the-head to discrete logarithms [37]. Despite not being useful for post-quantum signatures, this example inspired the present paper. For completeness, we recall the description of this discrete logarithm example in Section 4.

Using this as a source of inspiration, we describe the application of MPC-in-the-head to create efficient post-quantum signatures for isomorphism and group action problems. The idea of applying group actions in general for cryptographic purposes originates from [16]. Over the years, a large variety of problems compatible with this framework have been considered in cryptographic constructions. One of the most emblematic is probably the graph-isomorphism problem which already made an appearance in some of the seminal papers about zero-knowledge [31,6,32]. However, the literature contains many flavors of isomorphism and group action problems. Since not all are suitable for our purpose, we discuss isomorphisms and group actions in Section 3 and give precise requirements to fit them in our framework. Finally, in Section 5, we consider the application of MPC-in-the-head to create signatures from isomorphisms and group actions.

2 Notations and necessary primitives

Throughout the paper, $[a;b]$ denotes the set of all integers $x \in \mathbb{Z}$ such that $a \leq x \leq b$. The target security level of our constructions in bits is denoted by λ. We place ourselves in the random oracle model, and any hash function mentioned throughout this article should be considered a random oracle.

In our constructions, we also need puncturable pseudo-random functions (puncturable PRFs for short). A puncturable PRF family F on the set $[1;N]$ is a PRF family indexed by a key $\mathcal{K}$, that has domain $[1;N]$ and satisfies the following property:

– For all key $\mathcal{K}$ and index i, there exists a punctured key $\mathcal{K}_i^*$ together with an efficient evaluation algorithm $\mathcal{A}$ such that:

$$\forall j \in [1;N] \setminus \{i\} : \mathcal{A}(\mathcal{K}_i^*, j) = F_{\mathcal{K}}(j).$$

– Furthermore, given the punctured key $\mathcal{K}_i^*$, the value $F_{\mathcal{K}}(j)$ should remain indistinguishable from a random value.

As noticed in [13,38,15], a standard construction of puncturable PRFs can be derived from the famous tree-based PRF introduced in [30]. In this construction, a punctured key is made of the siblings of all nodes in the path from the punctured evaluation point to the root. In this construction, punctured keys are larger than the master key by a factor $\lceil \log_2 N \rceil$. Any construction with more compact punctured keys would directly increase the performance of the algorithms from Section 5.

3 Isomorphisms and group actions

From an abstract point of view, isomorphisms and group actions fit in a similar framework. In this framework, we have a finite set of objects $\mathbb{O}$ and a finite set of maps $\mathbb{S}$ from $\mathbb{O} \cup \{\perp\}$ to $\mathbb{O} \cup \{\perp\}$, where $\perp$ is a special symbol denoting an invalid object. This invalid object is useful since, in general, we do not require that every map applies to every object. Thus, to simplify notations, when ϕ is undefined at $\mathcal{O}$, we let instead $\phi(\mathcal{O}) = \perp$. We also define $\phi(\perp) = \perp$ for every map in $\mathbb{S}$. For compactness, we write $\mathbb{O}^\perp$ to denote $\mathbb{O} \cup \{\perp\}$. We require that the identity map Id on $\mathbb{O}^\perp$ belong to $\mathbb{S}$ and that $\mathbb{S}$ is closed under composition, i.e., for all ϕ_1 and ϕ_2 in $\mathbb{S}$, the composition $\phi_1 \circ \phi_2$ belongs to $\mathbb{S}$. Since the composition of maps is associative, $(\mathbb{S}, \circ)$ forms a monoid.

Given two objects $\mathcal{O}$ and $\mathcal{O}'$ in $\mathbb{O}^\perp$, we say that there are isomorphic if and only if there exist two maps in ϕ and ϕ' in $\mathbb{S}$ such that:

$$\phi(\mathcal{O}) = \mathcal{O}' \quad \text{and} \quad \phi'(\mathcal{O}') = \mathcal{O}.$$

From the above definition, we see that $\perp$ is isomorphic to itself and to no valid object from $\mathbb{O}$. In the special case where $(\mathbb{S}, \circ)$ is a group – instead of just a monoid – we denote the inverse map of ϕ by ϕ^{-1}. Note that in this case, the definition of $\mathcal{O}$ and $\mathcal{O}'$ being isomorphic simplifies to the existence of a map ϕ such that $\phi(\mathcal{O}) = \mathcal{O}'$. Indeed, ϕ' can be replaced by ϕ^{-1}. If, in addition, every map from $\mathbb{S}$ is defined on every object in $\mathbb{O}$, i.e., does not send any object to $\perp$, we are in a special case, which we refer to as being the case of a **group action**. To be consistent with the notations of[12], we define the group action law $*$ from $\mathbb{S} \times \mathbb{O}$ to $\mathbb{O}$ to be evaluation, i.e., we define $\phi * \mathcal{O} = \phi(\mathcal{O})$.

We can remark that a group action arising from our definition is naturally faithful. Indeed, the only $\phi \in \mathbb{S}$ such that for all $\mathcal{O} \in \mathbb{O}$ we have $\phi(\mathcal{O}) = \mathcal{O}$ is the identity map *Id*. However, it is not necessarily transitive, and $\mathbb{O}$ might be a disjoint union of more than one orbit. Remark that two objects are isomorphic if and only if they belong to the same orbit.

In all cases, we want to do fast computations, and we require the existence of an efficient algorithm that, given a pair $(\phi, \mathcal{O})$ outputs either $\phi(\mathcal{O})$ or $\perp$ when the map is incompatible with the given object. We also want multiplications and inversions in $\mathbb{S}$ to be efficiently computable and random elements in $\mathbb{S}$ to be efficiently samplable. On the flip side, we need some hardness assumption, i.e., given $\mathcal{O}$ and $\mathcal{O}'$, it should be hard to find a map from one to the other or even to decide about the existence of such a map. Finding a map between $\mathcal{O}$ and $\mathcal{O}'$ is usually called the **vectorization problem** (see [20]).

To make the sampling procedure in $\mathbb{S}$ explicit, we assume that the cardinality of $\mathbb{S}$ is known and that there exists an efficient (bijective) map γ from $\mathbb{Z}_{|\mathbb{S}|}$ to $\mathbb{S}$. This allows us to sample random group elements by evaluating $\gamma(r)$ where r is uniformly random in $\mathbb{Z}_{|\mathbb{S}|}$. We do not require that the inverse of γ is efficiently computable, but neither do we assume that γ is one-way.

When all the above efficiency constraints are satisfied, we say that we have a **computable group action**.

In fact, most isomorphism problems can be viewed as examples of computable group actions. Another well-known example of group actions comes from isogeny-based systems where a class group acts on a set of (isogenous) elliptic curves. Unfortunately, not all isogeny-based systems lead to computable group actions. Instead, they might induce a reduced form of computable group action where only some group elements can be applied efficiently.

3.1 Examples of isomorphisms and group actions

Graph isomorphisms Since graph isomorphisms were considered in the context of ZK protocols for very long [31,6,32], it is very natural to consider this problem as our first example. Remember that an undirected graph G on n vertices can be described by labeling the vertices from 1 to n and providing a symmetric indicator function E_G from $[1;n]^2 \rightarrow \{0,1\}$ such that $E_G(i,j) = 1$ if and only if there is an edge between the vertices i and j. Given a permutation σ in S_n the permutation group on $[1;n]$, we can define the permuted graph $\sigma(G)$ with indicator function defined by $E_{\sigma(G)}(i,j) = E_G(\sigma^{-1}(i), \sigma^{-1}(j))$. This provides our first example of a computable group action, where $\mathbb{O}$ is the set of all graphs on n vertices and $\mathbb{S}$ the symmetric group S_n on n elements. This example is efficiently computable since it is easy to apply, compose, and inverse permutations. Note that the set of graphs $\mathbb{O}$ comprises many distinct orbits; thus, this group action is not transitive.

Unfortunately, the graph isomorphism problem is solvable in quasi-polynomial time [2] and can no longer be a good candidate for cryptography.

Isomorphism of multivariate polynomials This isomorphism problem was proposed by Patarin in [42]. Given a finite field $\mathbb{F}_q$, the problem involves vectors of m polynomials over the polynomial ring in n variables $\mathbb{F}_q[x_1, x_2, \cdots, x_n]$ of total degree $D > 1$. The quadratic case where $D = 2$ is considered the most interesting for practical purposes.

Let F be such a vector of m polynomials and let A and B be matrices respectively from $GL_n(\mathbb{F}_q)$ and $GL_m(\mathbb{F}_q)$. We can create a new vector of polynomials:

$$G = B \times F(A\,(x_1, x_2, \cdots, x_n)^t),$$

i.e., we first multiply A with the column vector containing the x, evaluate F at these coordinates and multiply B with the result of this evaluation. The resulting vector also contains m polynomials of degree D in the n variables. Remark that these transformations are easy to apply, compose and invert.

We thus obtain a computable group action by letting $\mathbb{O}$ be the set of all vectors of m polynomials of total degree D in n variables and $\mathbb{S}$ be the product group $GL_n(\mathbb{F}_q) \times GL_m(\mathbb{F}_q)$.

Code equivalence Code equivalence is a well-known concept in coding theory. It considers the action of isometries (in the relevant metric) on codes. For simplicity, let us consider the case of linear binary code in the Hamming metric. Such a code $\mathcal{C}$ is a vector subspace of dimension k in the ambient space $\mathbb{F}_2^n$. It can be either described by a generator matrix G (of dimension $n \times k$) or a parity check matrix H (of dimension $(n-k) \times n$).

For our group action, we take for $\mathbb{O}$ the set of all such linear binary codes (with k and n fixed). To form the group $\mathbb{S}$, we take all isometries $\mathbb{F}_2^n$. In the Hamming metric, the isometries are simply the permutations of coordinates. Thus, as in the case of graph isomorphism, we have $\mathbb{S} = S_n$, and the group action is easily computable.

However, there is an essential consideration for the security of code equivalence. Indeed, if we directly let a permutation σ act on a parity check matrix (by permuting its columns), we obtain a problem that is easy to solve by simple inspection. To avoid this issue and obtain a hard problem, the parity check matrix needs to be transformed.

Remember that, using the parity check description, a (column) vector x belongs to $\mathcal{C}$ if and only if $H\,x = 0$. For a fixed code $\mathcal{C}$, there are many equivalent parity check matrices: if H is one of them, then the others are given by $A\,H$, where A is an arbitrary matrix from $GL_{n-k}(\mathbb{F}_2)$. To prevent the above attack, one can multiply H by a random matrix of this form (before or after applying the permutation). Another common approach is to put the parity check matrix in *systematic form.*

Cryptographic applications of code equivalence have been considered, for example, in [43,4,3,5,22]. Two of the candidates in NIST's current standardization effort for post-quantum signature are based on code equivalence in the Hamming and Rank metrics: LESS and MEDS. After the first version of the present paper was made available, an MPC-in-the-head signature based on code equivalence was proposed in [14].

Tensor isomorphisms The notion of tensor products is a very general algebraic construction that stems from multilinear products (see [39, Chapter XVI]). The notion of tensor isomorphism is very powerful, and many other isomorphism problems can be embedded into tensor isomorphisms [33]. It was proposed for cryptographic purposes in [36]. Since isomorphisms of tensors come from the action of matrix groups, they easily fit in our framework – at least when finite dimensional tensors over finite fields are considered.

Signature schemes have, in particular, been derived from trilinear forms, which are a specific type of tensors, see [44,23]. In that case, the choice of parameters must be careful to avoid the attack from [8]. The ALTEC submission in the additionnal NIST signature standardization round is based on this hard problem.

Lattice isomorphims The lattice isomorphism problem [34,24] is close to our framework but does not entirely fit since it considers an infinite group acting on an infinite set. Despite this, it is possible to design a zero-knowledge proof of knowledge of an isomorphism in this context [24], which is very similar to the protocols which are used for other isomorphism problems and presented in Section 4. For simplicity, we do not consider this case of infinite groups in our constructions. Some aspects of the security of lattice isomorphism problems are further discussed in [17].

Isogenous elliptic curves The idea of using elliptic curves and isogenies as group actions in cryptography originates in [21]. Mathematically, it is a perfect example since we have an abstract group (a class group) acting on a set of isogenous elliptic curves (see [29] for a survey). It stems from deep algebraic geometry, and the fact that it can be used in cryptography is fascinating. However, this depth can also hide unexpected avenues for attacks, as illustrated by the recent break of SIDH [19] .

Another difficulty with isogeny-based cryptography is computational. Indeed, only some isogenies can be computed efficiently, for example, low-degree isogenies, Frobenius, or multiplication by a constant. As a consequence, in general, it takes a lot of work to implement the full group action effectively. It can still be done once the class group structure has been fully computed, as done, for example, in [11]. To model this, [1] introduced the idea of restricted group actions. For a recent survey on using isogenies in knowledge proofs, the reader should consult [9].

4 State of the art

4.1 Standard signatures from group actions

The following schemes to perform identification from group actions are standard practice. In particular, they are presented in the technical overview Section of [12].

Basic identification scheme for group action. In this setting, the prover $\mathcal{P}$ has a public key consisting of two isomorphic objects $\mathcal{O}_0$ and $\mathcal{O}_1$ from $\mathbb{O}$. He also knows, as his private key, an isomorphism between $\mathcal{O}_0$ and $\mathcal{O}_1$. Depending on the proof system we want to use, it can be more convenient to know this isomorphism as a group element $\phi \in \mathbb{S}$ such that either $\mathcal{O}_0 = \phi * \mathcal{O}_1$ or $\mathcal{O}_1 = \phi * \mathcal{O}_0$. Of course, these two options are equivalent: it suffices to invert ϕ to switch from one choice to the other.

In this section, the private key is a group element $\phi \in \mathbb{S}$ such that $\mathcal{O}_0 = \phi * \mathcal{O}_1$. However, for other protocols, we may change the notation to clarify the exposition of protocols.

To prove his knowledge of ϕ, the prover can use the following protocol with soundness error 1/2.

- The prover $\mathcal{P}$ selects a random group element $\phi_r \in \mathbb{S}$ and outputs $\mathcal{O}' = \phi_r * \mathcal{O}_0$.
- The verifier $\mathcal{V}$ produces a random bit $b \in \{0, 1\}$.
- If $b = 1$, the prover $\mathcal{P}$ answers with $\phi' = \phi_r \circ \phi$. If $b = 0$, the prover returns $\phi' = \phi_r$.
- The verifier $\mathcal{V}$ checks that $\phi' * \mathcal{O}_b = \mathcal{O}'$ and accepts when the check succeeds. Otherwise, he rejects.

Identification scheme with long key. To improve the soundness, the public key can be replaced by a longer sequence of K isomorphic objects $(\mathcal{O}_i)_{i=0}^{K-1}$. In that case, the prover holds as private key a sequence of $K-1$ group elements $(\phi_i)_{i=1}^{K-1}$, such that for all i: $\mathcal{O}_0 = \phi_i * \mathcal{O}_i$. The following adapted protocol now provides soundness error $1/K$:

- The prover $\mathcal{P}$ selects a random group element $\phi_r \in \mathbb{S}$ and outputs $\mathcal{O}' = \phi_r * \mathcal{O}_0$.
- The verifier $\mathcal{V}$ produces a random query $c \in [0; K-1]$.
- If $c \neq 0$, the prover $\mathcal{P}$ answers with $\phi' = \phi_r \circ \phi_c$. If $c = 0$, the prover returns $\phi' = \phi_r$.
- The verifier $\mathcal{V}$ checks that $\phi' * \mathcal{O}_c = \mathcal{O}'$ and accepts when the check succeeds. Otherwise, he rejects.

Signature scheme. Using the Fiat-Shamir heuristic on a sufficient number of parallel repetitions of one of the two identification schemes yields a classical signature scheme. A lower bound on the size of such signatures has been studied in [12].

4.2 MPC-in-the-head and signature schemes

The MPC-in-the-head paradigm has recently been used to design efficient signature schemes based on problems such as syndrome decoding, multivariate polynomials, permuted kernel, ... The recent literature is abundant [7,27,25,28,26,41,18].

Since the practical use of MPC-in-the-head for signatures is quite recent, its newly gained relevance in the field can be measured by considering the recent NIST call for additionnal signatures. The first round features 40 candidates. Out of these, 7 are using MPC-in-the-head.

4.3 MPC-in-the-head for discrete logarithms

At CBCrypto'23, it was shown [37] that MPC-in-the-head can also be used to create signatures based on discrete logarithms. However, these signatures are less efficient than standard methods like Schnorr's signature scheme. Furthermore, using MPC-in-the-head does not prevent Shor's algorithm from breaking discrete logarithms and does not lead to post-quantum secure signatures. Thus, the main interest of the approach is pedagogical. Indeed, the multi-party protocol for proving discrete logarithm is straightforward, and when studying the resulting scheme, one can focus on the *in-the-head* aspect.

Basic multiparty protocol for discrete logarithms. Let $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N$ be a group of N provers that possess an additive sharing of the discrete logarithm x of an element $y = g^x$ in the cyclic group of order q generated by g. Thus, each party $\mathcal{P}_i$ holds a share x_i. If the sharing is correct, we should have:

$$x = \sum_{i=1}^{N} x_i \pmod{q}.$$

The parties want to run a protocol to verify that the sharing is indeed correct. We assume that all parties are honest but curious. In this context, we can verify the correctness of the sharing while keeping x secret even if $N-1$ parties try to learn it by putting their shares together.

The protocol is as follows:

- Each party i outputs $y_i = g^{x_i}$.

– Everyone check that $\prod_{i=1}^{N} y_i = y$, and accordingly accept or reject.

The protocol is correct since everyone accepts when the shares are valid, and the parties are honest.

If the sharing is performed correctly (uniformly at random), then a single missing share is enough to hide x completely. Furthermore, the protocol is genuinely $(N-1)$-private. Indeed, the joint view of the protocol outputs and $N-1$ shares can be easily simulated. It suffices to distribute random shares to all parties except one cheating i^*, who outputs $y_{i^*} = y/\prod_{j\neq i^*} y_j$ – after seeing the outputs of the other parties. This leads to a global transcript that follows the same distribution as a correct execution.

Identification protocol using MPC-in-the-head. The above multiparty protocol is easily transformed into an interactive two-party zero-knowledge proof of knowledge of x. In the simplest version, the prover $\mathcal{P}$ creates a fresh random sharing $x = \sum_{i=1}^{N} x_i \pmod q$, runs the above multiparty protocol and sends all outputs y_i to the verifier $\mathcal{V}$.

Then, $\mathcal{V}$ randomly selects an index $i^* \in [1; N]$ and send it to the prover. Then $\mathcal{P}$ returns $(x_j)_{j\neq i^*}$. Finally, $\mathcal{V}$ checks that $y_j = g^{x_j}$ for all positions except i^* and that $y = \prod_{i=1}^{N} y_i$.

A cheating prover $\mathcal{P}^*$ cannot create a valid sharing (or he would be able to learn x); thus, he must cheat in at least one position. This only goes undetected when the cheating position equals the query i^*. As a consequence, the protocol has a soundness error of $1/N$.

Reducing the communication. In the simple protocol, there are two big messages, namely $(y_i)_{i=1}^{N}$ and $(x_j)_{j\neq i^*}$. The first message is easily replaced by a short commitment, where $\mathcal{P}$ only sends $h = H(y_1||y_2||\cdots||y_N)$. At the final step, the verifier checks h as follows: first by computing all $y_j = g^{x_j}$ for $j \neq i^*$, then by letting $y_{i^*} = y/\prod_{j\neq i^*} y_j$. From this, $\mathcal{V}$ can recompute his own hash h' and check that $h = h'$. When the check is correct, one of two things must have happened:

– The reconstructed values y_i are the same as the original ones, and the compressed protocol emulates the simple one.
– The reconstructed values are different, and we have obtained a collision on the random oracle H.

To compress the second message $(x_j)_{j\neq i^*}$, we change the initial sharing to make use of a puncturable PRF. In this case, $\mathcal{P}$ chooses a random key $\mathcal{K}$ and let $x_i = \mathrm{PRF}_{\mathcal{K}}(i)$. Of course, these pseudo-random shares rarely sum to the target value x. To compensate for this, we compute an offset Δ_x such that:

$$x = -\Delta_x + \sum_{i=1}^{N} x_i \pmod q.$$

The first message sent now contains both the commitment h and the offset Δ_x. When receiving the query i^*, the proof sends back the punctured key $\mathcal{K}_{i^*}$. From this, $\mathcal{V}$ obtains all x_j for $j \neq i^*$ and can recompute the corresponding y_j. For the final value, he modifies the computation to account for Δ_x and computes:

$$y_{i^*} = \frac{y\, g^{\Delta_x}}{\prod_{j\neq i^*} y_j}.$$

Introducing the hypercube. The hypercube idea for MPC-in-the-head has been introduced in [41]. When applied to the above discrete logarithm instantiation, it can reduce the number of exponentiations needed while keeping the other relevant parameters of the identification protocol fixed. To make this explicit, let's assume that N is a power of two, namely $N = 2^n$. We denote the i-th bit of the binary decomposition of a number j by $B_i(j)$. The hypercube idea is to transform the initial N-wise sharing of x into n independent pairwise sharings. Indeed, for any $i \in [0; n-1]$ we can see that:

$$x = -\Delta_x + \sum_{B_i(j)=0} x_j + \sum_{B_i(j)=1} x_j \pmod q,$$

where only one of the two sums is known to a verifier that lacks one value in position i^*.

Instead of committing to the N values y_i as before, the prover instead commits to the $2n$ values

$$g^{\sum_{B_i(j)=b} x_j},$$

with $b \in \{0,1\}$ and $i \in [0; n-1]$. Since additions are much faster to compute than exponentiations, this opens the choice of parameters to include larger values of N; thus, reducing the number of necessary repetitions in the derived signature scheme.

5 Introducing MPC-in-the-head for group actions

In this section, we want to improve the standard identification protocols from Section 4.1. We use similar notations but shift the value of K by one since this simplifies the expression of the soundness error in our protocols. We now have a public key consisting of a sequence of $K+1$ isomorphic objects $(\mathcal{O}_i)_{i=0}^{K}$. The prover knowns as corresponding private key a sequence of K group elements $(\phi_i)_{i=1}^{K}$, such that for all $i \in [1; K] : \mathcal{O}_i = \phi_i * \mathcal{O}_0$.

We mentioned in Section 4.1 that there were two options for defining the isomorphisms. We are now using the alternative option with $\mathcal{O}_0$ on the right since it simplifies the writing of the protocol.

Note that $\mathcal{O}_0$ does not need to be counted in the public key size. Indeed, depending on the preferences of the designers, it can be a system-wide parameter or a pseudo-random element given in compressed form by a seed. This is an additional argument in favor of offsetting the definition of K by 1.

We can closely follow the strategy from Section 4.3.

Basic multiparty protocol for a group action. We temporarily assume that $K = 1$ and want to share and verify the knowledge of the group element ϕ such that $\mathcal{O}_1 = \phi * \mathcal{O}_0$. For this, we give to each of the N provers $\mathcal{P}_i$, with $i \in [1; N]$, a random group element $\phi^{(i)}$. We also create and publish an additional offset map ϕ^{Δ}, such that:

$$\phi^{\Delta} \circ \phi^{(N)} \circ \phi^{(N-1)} \circ \cdots \circ \phi^{(1)} = \phi.$$

Since the protocol creates extra intermediate objects, to help distinguish them from objects in the public key, we denote them using the symbol Θ instead of $\mathcal{O}$. To simplify the writing of the protocol, we define the first of these intermediate objects as $\Theta^{(0)} = \mathcal{O}_0$. The discrete logarithm protocol can now be adapted to:

- Each party i waits for $\Theta^{(i-1)}$, then computes and outputs $\Theta^{(i)} = \phi^{(i)} * \Theta^{(i-1)}$.

– Everyone check that $\phi^{\Delta} * \Theta^{(N)} = \mathcal{O}_1$, and accordingly accept or reject.

We get the same correctness, hiding, and zero-knowledge properties. However, a simulator wanting to cheat on a single party i^* proceeds slightly differently. All parties before i^* follow the normal protocol, and every party after i^* works backward. Namely, $\Theta^{(N)}$ is computed as $(\phi^{\Delta})^{-1} * \mathcal{O}_1$ and each $\Theta^{(j)}$ for $i^* \leq j < N$ as $\Theta^{(j)} = (\phi^{(j+1)})^{-1} * \Theta^{(j+1)}$.

Identification protocol from group action. Since the adaptation is quite straightforward, we skip the presentation of the simple protocol. Instead, we directly work with a public key of size K and integrate the compression techniques based on commitment and puncturable PRFs.

– The prover $\mathcal{P}$ chooses a random key $\mathcal{K}$ for the puncturable PRF and sets $\phi^{(i)} = \gamma(\mathrm{PRF}_{\mathcal{K}}(i))$. Remember that γ (introduced in Section 3) converts the output of the PRF into a group element. Then, $\mathcal{P}$ sets $\Theta^{(0)} = \mathcal{O}_0$ and for all i in $[1; N]$ computes $\Theta^{(i)} = \phi^{(i)} * \Theta^{(i-1)}$. He then commits by sending the hash value:

$$h = H(\Theta^{(1)} || \Theta^{(2)} || \cdots || \Theta^{(N)}).$$

– The verifier $\mathcal{V}$ sends as query a pair (i^*, k), with $i^* \in [1; N]$ and $k \in [1; K]$.
– The prover returns the punctured key $\mathcal{K}_{i^*}$ and the offset map ϕ^{Δ_k} that satisfies:

$$\phi^{\Delta_k} \circ \phi^{(N)} \circ \phi^{(N-1)} \circ \cdots \circ \phi^{(1)} = \phi_k.$$

– The verifier can now compute all the objects $\Theta^{(i)}$ by a forward computation from $\mathcal{O}_0$ up to $i^* - 1$ and by a backward computation from $\mathcal{O}_k$ from N down to i^*. He then recomputes and checks the hash commitment h.

This protocol has a soundness error of $1/(NK)$.

Note 1. In fact, it is possible for the verifier to choose between $NK + 1$ options instead of NK. Indeed, there is the extra option of giving out the complete key of the PRF without revealing any of the offset maps. Since this option requires a more compact answer, it might even be advantageous in terms of size to use it more often than the other options.

Note 2. It is also possible to further increase the number of questions to $N\,K\,(K+1)$ or $N\,K\,(K+1) + 2\,K + 2$ by putting an offset map on both sides of the chain of maps. With this change, the verifier can ask for a chain from $\mathcal{O}^{(i)}$ to $\mathcal{O}^{(j)}$ with a single missing link (out of N positions), which gives him $N\,K\,(K+1)$ choices. If we also permit the missing link to be one of the two offset maps on the right or left, we get $2(K+1)$ extra possibilities.

Corresponding signature scheme. Using the Fiat-Shamir heuristic with r repetitions, we get an overall security of the resulting signature of $\lambda = r \log_2(NK)$. The public key size corresponds to the size of K objects from $\mathbb{O}$. The size of the private key can be minimal since a seed to generate all maps $(\phi_k)_{k=1}^{K}$ pseudo-randomly suffices.

The signature contains the following elements:

– A global commitment to the r rounds simultaneously, i.e., one hash output.
– One punctured PRF key and one offset map ϕ^{Δ} (group element) for each round.

To evaluate the bit-size of such a signature, we first need to set the hash function's output size and the puncturable PRF's size consistently with the desired security level. Concerning the hash function, we want to avoid collisions and thus need outputs on 2λ bits. For the puncturable PRF, if we use the GGM construct, it might seem at first that we also want to avoid collisions and need 2λ bits for regular keys and $2\lambda \log_2(N)$ bits for a punctured key. However, it is also possible to apply the standard trick of using a global random salt on at least λ bits and reduce the PRF key sizes to λ and $\lambda \log_2(N)$ bits.

Similarly, the bit-size of group elements must be at least 2λ to avoid generic attacks on the group action. With $K = 1$, the signature size is easy to compute, assuming we are using r rounds and N parties. The global committment costs at least 2λ bits and the salt costs at least λ bits. Each round requires one group element, i.e., 2λ bit, and a punctured key of at least $\lambda \log_2(N)$ bits. Thus, we obtain a total of at least

$$3\lambda + r\lambda \log_2(N) + 2r\lambda \approx \lambda^2 + (2r+3)\lambda \text{ bits},$$

recalling that $r \log_2(N) \approx \lambda$. It is interesting to realize that N vanishes from the final expression we obtain.

Note on the hypercube technique. With group actions, the hypercube technique does not offer the same bonuses as with discrete logarithms. First, taking different regroupings on the hypercube will only lead to the same offset maps if the group is abelian. Transmitting a different offset map for every possible regrouping would remove any advantage we could gain from the hypercube. Even with a commutative group, since it is not certain that group operations are less costly than group actions, the advantage of the hypercube technique is unclear. However, if γ turns out to be a group morphism from an additive group with fast operations, using hypercubes would be very positive. Unfortunately, most group actions do not belong to this exceptional case.

Comparison with the standard technique. Compared to the technique described in Section 4.1, using MPC-in-the-head gives better soundness per round, especially when a small public key is used. However, each round requires the computation of many group actions (or isomorphisms). Thus we can get a smaller size at the cost of more computations. The signature sizes can also be better since the punctured PRF keys are, in general, more compact to represent than arbitrary group elements.

6 Punctured PRFs in the standard approach

Puncturable PRFs can also optimize the classical approach described in Section 4.1. As far as we know, this idea was first introduced in [10] using the name *seed tree* instead of puncturable PRF. The resulting approach is similar to the one developed in Section 5. However, viewing it as an instantiation of MPC-in-the-head is no longer possible.

To allow for direct comparison with the MPC-in-the-head approach, let us fully describe a single round of the identification protocol with K elements in the public key, using similar notations. To simplify the description of the protocol, it is convenient to shift back to the same private keys as in Section 4.1, namely, for all i: $\mathcal{O}_0 = \phi_i * \mathcal{O}_i$.

- The prover $\mathcal{P}$ chooses a random key $\mathcal{K}$ for the puncturable PRF and sets $\phi^{(i)} = \gamma(\mathrm{PRF}_{\mathcal{K}}(i))$. Then, for all i in $[1; N]$, the prover sets $\Theta^{(i)} = \phi^{(i)} * \mathcal{O}_0$. He then commits by sending the hash value:
$$h = H(\Theta^{(1)} || \Theta^{(2)} || \cdots || \Theta^{(N)}).$$

– The verifier $\mathcal{V}$ sends as query a pair (i^*, k), with $i^* \in [1; N]$ and $k \in [1; K]$.
– The prover returns the punctured key $\mathcal{K}_{i^*}$ and the map $\phi' = \phi^{(i^*)} \circ \phi_k$.
– The verifier can now compute all the objects $\Theta^{(j)}$ with $j \neq i^*$ directly from $\mathcal{O}_0$ and $\Theta^{(i^*)}$ by applying ϕ' to $\mathcal{O}_k$. He then recomputes and checks the hash commitment h.

As with the MPC-in-the-head approach, we get soundness error $1/(NK)$ from sending one group element and one punctured key.

7 On the lower bound of [12]

In [12], Boneh, Guan, and Zhandry propose a lower bound on the size of signatures obtained for schemes arising from group actions. This proof uses Maurer's model [40] for group actions. An important technicality is that signatures are, in fact, impossible in Maurer's model. To sidestep this issue, the authors instead bound the communication size in the corresponding identification scheme, which is then converted into a signature using the Fiat-Shamir heuristic.

Using this approach, they give a lower bound on the number of set elements that need to be transmitted by the prover to the verifier. Their informal statement of the resulting theorem is:

Theorem 1 (Theorem 1 of [12]). *For any public-coin identification protocol secure against eavesdropping in a black box (potentially non-abelian) group action model, the sender must send at least* $(\lambda - 1)/\log_2 \lambda$ *set elements to achieve soundness error* $2^{-\lambda}$.

One can easily remark that using punctured PRFs as in Section 5 or 6 can reduce the number of elements sent below this bound. Indeed, even with a single element in the public key, using a punctured PRF with $N = \lambda^c$ outputs, we get a soundness error of N^{-r} using r rounds and thus sending r group elements. To achieve soundness error $2^{-\lambda}$, we can take $r = \lambda/(c \log_2 \lambda)$. As c can be arbitrary, this beats the bound of Theorem 1.

This comes from the fact that punctured PRFs do not fit in the security model of [12] and that an adversary could use brute force to recover the primary key from the punctured key, thus making the identification scheme insecure in Maurer's model.

Yet, when looking at concrete sizes, we see, following Section 5, that with this choice of parameters, the total number of bits that are transmitted is larger than λ^2, which is above the lower bound of [12] when expressed in bits.

However, if one could solve the open problem of designing a puncturable PRF with shorter punctured keys, possibly based on the use of the group action, the signatures obtained via puncturable PRFs would be significantly improved.

8 Formal schemes and their security proof

In this section, we formalize the intuitive descriptions of the schemes proposed in Sections 5 and 6, with a target security of λ bits. We consider two variants for each scheme, depending on the output size of the core random oracle $\mathcal{H}$ that we use in the constructions. The first variant considers a large output of 4λ bits and does not use salt. The second variant uses salt and considers a reduced output size of λ bits.

In both cases, we assume a group action with a group of size close to $2^{2\lambda}$ to preclude generic attacks faster than 2^{λ}. With the random oracle now defined, we can describe the unsalted and salted puncturable PRFs we consider. In the unsalted version, every node of the tree in the

```
Reject::Global variable initialy set to false

HlogBook::Global dictionary initially empty (Memory for the random oracle)
--------------------------------------------------------------------------

Function $\mathcal{H}_u$(Mess::bitstring) :: $4\lambda$-bitstring
 If Mess appears as an index in HlogBook
  Return HlogBook[Mess]
 Else
  Let Value be a uniformly random $4\lambda$-bitstring
  If Value already appears as a value in HlogBook
   Set the global variable Reject to true
  End If
  Define HlogBook[Mess] = Value
  Return Value
 End If
End Function
```

Table 1. Pseudo-code of the Random Oracle for the unsalted case

GGM construct has 4λ bits. This is large enough to convert tree leaves into a statistically close to uniform distribution on the group by reducing the value of each leaf modulo the group size before applying the map γ to convert it to a group element. In the salted version, we use a salt of size 3λ and internal nodes of size λ in the GGM tree. For the leaves, we reexpand to 4λ bits before converting to a group element.

```
HlogBook::Global dictionary initially empty (Memory for the random oracle)
--------------------------------------------------------------------------

Function $\mathcal{H}_s$(Mess::bitstring) :: $\lambda$-bitstring
 If Mess appears as an index in HlogBook
  Return HlogBook[Mess]
 Else
  Let Value be a uniformly random $\lambda$-bitstring
  Define HlogBook[Mess] = Value
  Return Value
 End If
End Function
```

Table 2. Pseudo-code of the Random Oracle for the salted case

```
Function Descendant (Level::integer,Node::$4\lambda$-bitstring, SubPath::bitstring) :: $4\lambda$-bitstring
  Return $\mathcal{H}_u$("PRFu"‖Level‖Node‖SubPath)
End Function
```

```
Function RecursDescent (Level::integer, Node::$4\lambda$-bitstring, Path::bitstring) :: $4\lambda$-bitstring
  If Level is equal to the length of Path
    Return Node
  Else
    Let SubPath = Path[$1 \ldots$ Level + 1]
    Let NextNode = Descendant(Level, Node, SubPath)
    Return RecursDescent(Level + 1, NextNode, Path)
  End If
End Function
```

```
Function LeafToGroup (Node::$4\lambda$-bitstring):: GroupElt
  Let Value = ToInteger(Node)   Return $\gamma$(Value mod $|\mathbb{S}|$)
End Function
```

```
Function GroupEltFromKey (RootKey::$4\lambda$-bitstring, PathToLeaf::bitstring):: GroupElt
  Return LeafToGroup(RecursDescent(0, RootKey, PathToLeaf))
End Function
```

```
Function PunctureKey (RootKey::$4\lambda$-bitstring, Path::bitstring):: array of bitstring
  Init PuncturedKey (as empty array)
  Set PuncturedKey[0] = Path
  For Level from 1 to TreeDepth
    Set CurrentPath = Path[$1 \ldots$ Level]
    Flip Bit CurrentPath[Level]
    Set PuncturedKey[Level] = RecursDescent(0, RootKey, CurrentPath)
  End For
  Return PuncturedKey
End Function
```

```
Function GroupEltFromPuncKey (PuncturedKey, PathToLeaf::bitstring):: GroupElt
  Set ForbiddenPath = PuncturedKey[0]
  If Path is equal to ForbiddenPath
    Return $\perp$
  End If
  Let Level be the first bit position where Path differs from ForbiddenPath
  Let Leaf = RecursDescent(Level, PuncturedKey[Level], Path)
  Return LeafToGroup(Leaf)
End Function
```

Table 3. Pseudo-code of the GGM puncturable PRF for the unsalted case

```
Function Descendant (Level, Salt, Node, SubPath) :: λ-bitstring
 Return $\mathcal{H}_s$("PRFs"‖Level‖Salt‖Node‖SubPath)
End Function
```

```
Function RecursDescent (Level, Salt, Node, Path) :: λ-bitstring
 If Level is equal to the length of Path
  Return Node
 Else
  Let SubPath = Path[1 . . . Level + 1]
  Let NextNode = Descendant(Level, Salt, Node, SubPath)
  Return RecursDescent(Level + 1, Salt, NextNode, Path)
 End If
End Function
```

```
Function LeafToGroup (Salt, Node):: GroupElt
 Let ExpandedNode = $\mathcal{H}_s$("Exp0"‖Salt‖Node)‖$\mathcal{H}_s$("Exp1"‖Salt‖Node)‖
                    $\mathcal{H}_s$("Exp2"‖Salt‖Node)‖$\mathcal{H}_s$("Exp3"‖Salt‖Node)
 Let Value = ToInteger(ExpandedNode)
 Return $\gamma$(Value mod $|\mathbb{S}|$)
End Function
```

```
Function GroupEltFromKey (Salt, RootKey, PathToLeaf):: GroupElt
 Return LeafToGroup(Salt, RecursDescent(0, Salt, RootKey, PathToLeaf))
End Function
```

```
Function PunctureKey (Salt, RootKey, Path):: array of bitstring
 Init PuncturedKey (as empty array)
 Set PuncturedKey[0] = Path
 For Level from 1 to TreeDepth
   Set CurrentPath = Path[1 . . . Level]
   Flip Bit CurrentPath[Level]
   Set PuncturedKey[Level] = RecursDescent(0, Salt, RootKey, CurrentPath)
 End For
 Return PuncturedKey
End Function
```

```
Function GroupEltFromPuncKey (Salt, PuncturedKey, PathToLeaf):: GroupElt
 Set ForbiddenPath = PuncturedKey[0]
 If Path is equal to ForbiddenPath
  Return $\perp$
 End If
 Let Level be the first bit position where Path differs from ForbiddenPath
 Let Leaf = RecursDescent(Level, Salt, PuncturedKey[Level], Path)
 Return LeafToGroup(Salt, Leaf)
End Function
```

Table 4. Pseudo-code of the GGM puncturable PRF for the salted case

With the explicit description of the puncturable PRFs in the salted and unsalted cases available, we can now give pseudo-code for the two identification protocols implementing the informal descriptions of Sections 5 and 6. To save space, we write the two protocols with an optional argument for the salt. Furthermore, both protocols have public coins for the verifier, so we do not need to give any code for the verifier query. Instead, we split the code of the prover in two phases. The first phase does not receive the verifier's query as input, while the second phase inherits the memory of the first phase and additionally receives the verifier's query. We also give the verification code that checks that the answers of the two phases are consistent with the public key and the query. For simplicity, we only formalize the description for the single key version where $K = 1$. The description of the identification protocol depends on the following elements:

– The target bit-security level λ.
– The number of leaves of the puncturable PRF, which we set as a power of two, $N = 2^t$.
– A group action of the group $\mathbb{S}$ on the set $\mathbb{O}$. To be consistent with the security level, we assume that $\mathbb{O}$ forms a single orbit of size approximately $2^{2\lambda}$. We also assume that the best algorithm that, given two random elements $\mathcal{O}$ and $\mathcal{O}'$, finds $\phi \in \mathbb{S}$ such that $\mathcal{O}' = \phi * \mathcal{O}$ has expected runtime $\Omega(2^\lambda)$.
– The starting point in $\mathbb{O}$ for any prover is the scheme-wide parameter $\mathcal{O}_0$.
– The private key of the prover $\mathcal{P}$ is a random element $\phi_{\mathcal{P}}$ in $\mathbb{S}$.
– The public key of $\mathcal{P}$ is $\mathcal{O}_{\mathcal{P}} = \phi_{\mathcal{P}} * \mathcal{O}_0$. This means that for the formalization of the protocol from Section 6, we verify the relation $\mathcal{O}_0 = \phi_{\mathcal{P}}^{-1} * \mathcal{O}_{\mathcal{P}}$.
– A commitment function that is implemented using the random oracle. In the unsalted case, the commitment to a message M is simply evaluating has $\mathcal{H}_u(\text{Commit}\|M)$. In the salted case, the direct adaptation of this commitment would be too short to prevent collisions, so we instead use $\mathcal{H}_s(\text{Commit0}\|M)\|\mathcal{H}_s(\text{Commit1}\|M)\|\mathcal{H}_s(\text{Commit2}\|M)$.

```
Function Phase_1:: Commitment and GroupElt
 Let RootKey be a uniform random bit string (of the expected size $4\lambda$ or $\lambda$)
 Let Salt be a uniform random bit string (of the expected size 0 or $3\lambda$)
 Let CommitString be the empty string
 Let $\Theta = \mathcal{O}_0$
 Let $\phi_\Delta = \phi_{\mathcal{P}}$
 For Index from 0 to $N - 1$
  Let Path be a $t$-bit binary encoding of Index
  Let $\phi_{\text{Index}} = \text{GroupEltFromKey}([\text{Salt}], \text{RootKey}, \text{Path})$
  Let $\Theta = \phi_{\text{Index}} * \Theta$
  Append Encoding of $\Theta$ to CommitString
 Let $\phi_\Delta = \phi_\Delta \circ \phi_{\text{Index}}^{-1}$
 End For
 Return Commit(CommitString) and $\phi_\Delta$
End Function
```

Table 5. Phase 1 of the identification protocol for the MPC-in-the-head

We can now state our main theorem about the two identification protocols in their unsalted and salted versions.

```
Function Phase_2(Query::[0...N-1]):: [Salt and] PuncturedKey
 Import RootKey [and Salt] from $\text{Phase}_1$
 Convert Query to a binary QueryPath
 Return [Salt ||] PunctureKey([Salt, ]RootKey, QueryPath)
End Function
```

Table 6. Phase 2 of the identification protocol for the MPC-in-the-head

```
Function Verify([Salt, ]PuncturedKey, Commitment, $\phi_\Delta$):: Boolean
 Let Query = ToInteger(PuncturedKey[0])
 Let $\Theta = \mathcal{O}_0$
 Let CommitPartA be the empty string
 For Index from 0 to Query $- 1$
  Let Path be a $t$-bit binary encoding of Index
  Let $\phi_{\text{Index}}$ = GroupEltFromKey([Salt], RootKey, Path)
  Let $\Theta = \phi_{\text{Index}} * \Theta$
  Append Encoding of $\Theta$ to CommitPartA
 End For
 Let CommitPartB be the empty string
 Let $\Theta = \phi_\Delta^{-1} * \mathcal{O}_\mathcal{P}$
 For Index from $N - 1$ downto Query $+ 1$
  Prepend Encoding of $\Theta$ to CommitPartB
  Let Path be a $t$-bit binary encoding of Index
  Let $\phi_{\text{Index}}$ = GroupEltFromKey([Salt], RootKey, Path)
  Let $\Theta = \phi_{\text{Index}}^{-1} * \Theta$
 End For
  Prepend Encoding of $\Theta$ to CommitPartB
 If Commit(CommitPartA||CommitPartB) is equal to Commitment
  Return true
 Else
  Return false
 End If
End Function
```

Table 7. Verification phase for the MPC-in-the-head identification protocol

```
Function Phase_1:: Commitment
 Let RootKey be a uniform random bit string (of the expected size $4\lambda$ or $\lambda$
 Let Salt be a uniform random bit string (of the expected size 0 or $3\lambda$
 Let CommitString be the empty string
 For Index from 0 to $N-1$
  Let Path be a $t$-bit binary encoding of Index
  Let $\phi_{\text{Index}} = \text{GroupEltFromKey}([\text{Salt}], \text{RootKey}, \text{Path})$
  Let $\Theta = \phi_{\text{Index}} * \mathcal{O}_0$
  Append Encoding of $\Theta$ to CommitString
 End For
 Return Commit(CommitString) and $\phi_\Delta$
End Function
```

Table 8. Phase 1 of the alternative identification protocol.

```
Function Phase_2(Query::[0...N-1]):: [Salt,] PuncturedKey and GroupElt
 Import RootKey [and Salt] from Phase$_1$
 Convert Query to a binary QueryPath
 Let $\phi_\Delta = \text{GroupEltFromKey}([\text{Salt}], \text{RootKey}, \text{QueryPath}) \circ \phi_{\mathcal{P}}^{-1}$
 Return [Salt ||] PunctureKey([Salt, ]RootKey, QueryPath) || $\phi_\Delta$
End Function
```

Table 9. Phase 2 of the alternative identification protocol

```
Function Verify([Salt, ]PuncturedKey, Commitment, $\phi_\Delta$):: Boolean
 Let Query = ToInteger(PuncturedKey[0])
 Let CommitPart be the empty string
 For Index from 0 to $N-1$
  If Index is equal to Query
   Let $\Theta = \phi_\Delta * \mathcal{O}_{\mathcal{P}}$
  Else
   Let $\Theta = \phi_{\text{Index}} * \mathcal{O}_0$
  End If
  Append Encoding of $\Theta$ to CommitString
 End For
 If Commit(CommitString) is equal to Commitment
  Return true
 Else
  Return false
 End If
End Function
```

Table 10. Verification phase of the alternative identification protocol

Theorem 2. *The unsalted and salted versions of the MPC-in-the-head and alternative group action based protocols described in this section are statistical honest verifier Zero-Knowledge proofs of knowledge with soundness error $1/N+\epsilon$, where ϵ is proportional to the inverse runtime of the best adversary that breaks the group action problem.*

Proof. **Completeness.** By simple inspection, we see that the verification always accepts the outputs given by a valid prover.

Statistical honest verifier Zero-Knowledge. We fist construct a simulator $\mathcal{S}$ that operates as follows:

- Pick (uniformly at random) a RootKey, a Salt (optionally), a map ϕ_Δ, and a Query.
- Create a PuncturedKey from Rootkey, Salt, and Query.
- Run the Verify function (without providing the Commitment value) to learn the expected Commitment.
- Return a simulated transcript built from PuncturedKey, Salt, Query, Commitment, and ϕ_Δ.

We want to prove that this output distribution is statistically indistinguishable from a valid transcript. To do that, we remark that if a valid prover runs with the same values for Rootkey, Salt, and Query, all the maps ϕ_{Index} that he creates for Index $\neq$ Query are already specified from the run of the simulator $\mathcal{S}$. Only ϕ_{Query} is missing. Furthermore, there is a unique map ϕ_{Query} that would produce the same transcript. This map can be forced by programming the random oracle at the relevant point (or points in the salted version). So the simulated transcript can also be produced by a legitimate prover.

Let's look at the probability of getting this specific transcript from the simulator and the prover. They only differ because ϕ_Δ is uniform for the simulator but obtained using LeafToGroup on a random value for the prover (followed by a composition). Since the output of LeafToGroup is statistically close to a random map, we see that the simulated distribution is statistically close to the real one.

Proof of Knowledge. Let $\mathcal{P}^*$ be a prover that convinces the verifier with probability $\geq (1/N)+\epsilon$, with a non-negligible value of ϵ. We show how to build an extractor, with read access to the random oracle memory, that learns the map $\phi_{\mathcal{P}}$.

We first remark that all the random oracle queries made by possibly many executions of the verify function on many (different) transcripts are distinct with overwhelming probability. In the salted version, it is clear that identical queries can only occur if identical salt values have been selected. Since these values have 3λ bits, the collision probability, even with 2^λ executions, is negligible. In the unsalted case, identical queries could only occur if the global variable Reject is set during the execution, which could only happen with a collision on values spanning 4λ bits.

After running the first phase of $\mathcal{P}^*$, the extractor can recover the input to the random oracle that produced the received commitment value. He can then parse this string into a sequence of objects $(\mathcal{O}^{(i)})_{i=0}^{N-1}$ Note that if the commitment has not been produced by calling the random oracle or if the parsing is incorrect, the verification process can only succeed with an exponentially small probability.

From the parsing, the extractor reconstructs a candidate (possibly incomplete) tree for the puncturable PRF. Since the reconstruction is slightly simpler for the alternative scheme, let us start with it. For every object $\mathcal{O}^{(i)}$, we search for all oracle queries whose input matches a call to the function Descendant corresponding to this leave. For each of them, we recompute the corresponding group element ϕ_{Cand} and check whether it matches $\mathcal{O}^{(i)} = \phi_{\text{Cand}} * \mathcal{O}_0$.

We memorize all the matching maps we can find. For the MPC-in-the-head case, we search for maps linking $\mathcal{O}^{(i)}$ to its expected predecessor. Three cases may occur; the extractor might reconstruct all maps, or miss precisely one, or miss more than one. In the third case, for any query that can be made, the second phase of $\mathcal{P}^*$ needs to provide a PuncturedKey that gives a valid group element on at least one missing position. This PuncturedKey has to use a fresh Oracle query to achieve this, which can only occur with negligible probability.

The cases of zero or one missing position behave differently with the two protocols. With the MPC-in-the-head authentication protocol, the following applies:

– If no position is missing then – including the offset map – we have a candidate path to go from $\mathcal{O}_0$ to $\mathcal{O}_{\mathcal{P}}$. If this path is valid, the extractor learns $\phi_{\mathcal{P}}$. If not, there is no answer that the second phase of $\mathcal{P}^*$ can give that would result in an accepting transcript.
– If one position is missing, the path from $\mathcal{O}_0$ to $\mathcal{O}_{\mathcal{P}}$ is incomplete. Furthermore, if the query sent to the second phase corresponds to the missing position, $\mathcal{P}^*$ could provide the correct PuncturedKey. However, this only accounts for a $1/N$ success probability. The extractor thus queries at another position and learns the complete path if the second phase successfully answers. This occurs with probability $\geq \epsilon$. Since a correct, complete path provides $\phi_{\mathcal{P}}$, combining the extractor with $\mathcal{P}^*$ breaks the group action problem in time $1/\epsilon$.

With the alternative protocol, the second phase of $\mathcal{P}^*$ cannot give a valid answer on a known position without revealing $\phi_{\mathcal{P}}$. This leads to the same conclusion that combining the extractor and $\mathcal{P}^*$ can break the group action problem in time $1/\epsilon$.

Note that because there is a generic group action solver with complexity 2^λ with our parameters, the value of ϵ specified in the statement of the theorem is large enough also to include the sheer luck successes where a fresh random oracle query matches the awaited value.

Since the signature schemes are derived using the Fiat-Shamir technique, giving a security proof of the protocols is sufficient. Indeed, in this setting, the security of the signature schemes directly follows from Theorem 2.

References

1. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 411–439. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_14
2. Babai, L.: Graph isomorphism in quasipolynomial time [extended abstract]. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 684–697. ACM Press (Jun 2016). https://doi.org/10.1145/2897518.2897542
3. Barenghi, A., Biasse, J.F., Ngo, T., Persichetti, E., Santini, P.: Advanced signature functionalities from the code equivalence problem. Cryptology ePrint Archive, Report 2022/710 (2022), `https://eprint.iacr.org/2022/710`
4. Barenghi, A., Biasse, J.F., Persichetti, E., Santini, P.: LESS-FM: Fine-tuning signatures from the code equivalence problem. In: Cheon, J.H., Tillich, J.P. (eds.) Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021. pp. 23–43. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-81293-5_2
5. Barenghi, A., Biasse, J.F., Persichetti, E., Santini, P.: On the computational hardness of the code equivalence problem in cryptography. Cryptology ePrint Archive, Report 2022/967 (2022), `https://eprint.iacr.org/2022/967`
6. Bellare, M., Micali, S., Ostrovsky, R.: Perfect zero-knowledge in constant rounds. In: 22nd ACM STOC. pp. 482–493. ACM Press (May 1990). https://doi.org/10.1145/100216.100283

7. Beullens, W.: Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 183–211. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45727-3_7
8. Beullens, W.: Graph-theoretic algorithms for the alternating trilinear form equivalence problem. Cryptology ePrint Archive, Report 2022/1528 (2022), `https://eprint.iacr.org/2022/1528`
9. Beullens, W., Feo, L.D., Galbraith, S.D., Petit, C.: Proving knowledge of isogenies – a survey. Cryptology ePrint Archive, Paper 2023/671 (2023), `https://eprint.iacr.org/2023/671`, `https://eprint.iacr.org/2023/671`
10. Beullens, W., Katsumata, S., Pintore, F.: Calamari and Falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 464–492. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_16
11. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 227–247. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-34578-5_9
12. Boneh, D., Guan, J., Zhandry, M.: A lower bound on the length of signatures based on group actions and generic isogenies. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 507–531. Springer (2023). https://doi.org/10.1007/978-3-031-30589-4_18
13. Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 280–300. Springer, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42045-0_15
14. Borin, G., Persichetti, E., Santini, P.: Zero-knowledge proofs from the action subgraph. Cryptology ePrint Archive, Paper 2023/718 (2023), `https://eprint.iacr.org/2023/718`, `https://eprint.iacr.org/2023/718`
15. Boyle, E., Goldwasser, S., Ivan, I.: Functional signatures and pseudorandom functions. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 501–519. Springer, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-642-54631-0_29
16. Brassard, G., Yung, M.: One-way group actions. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO'90. LNCS, vol. 537, pp. 94–107. Springer, Heidelberg (Aug 1991). https://doi.org/10.1007/3-540-38424-3_7
17. Budroni, A., Chi-Domínguez, J.J., Kulkarni, M.: Lattice isomorphism as a group action and hard problems on quadratic forms. Cryptology ePrint Archive, Paper 2023/1093 (2023), `https://eprint.iacr.org/2023/1093`, `https://eprint.iacr.org/2023/1093`
18. Carozza, E., Couteau, G., Joux, A.: Short signatures from regular syndrome decoding in the head. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 532–563. Springer (2023). https://doi.org/10.1007/978-3-031-30589-4_19
19. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 423–447. Springer (2023). https://doi.org/10.1007/978-3-031-30589-4_15
20. Castryck, W., Meeren, N.V.: Two remarks on the vectorization problem. Cryptology ePrint Archive, Report 2022/1366 (2022), `https://eprint.iacr.org/2022/1366`
21. Couveignes, J.M.: Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291 (2006), `https://eprint.iacr.org/2006/291`
22. D'Alconzo, G.: Monomial isomorphism for tensors and applications to code equivalence problems. Cryptology ePrint Archive, Paper 2023/396 (2023), `https://eprint.iacr.org/2023/396`, `https://eprint.iacr.org/2023/396`

23. D'Alconzo, G., Gangemi, A.: TRIFORS: LINKable trilinear forms ring signature. Cryptology ePrint Archive, Report 2022/1170 (2022), `https://eprint.iacr.org/2022/1170`
24. Ducas, L., van Woerden, W.P.J.: On the lattice isomorphism problem, quadratic forms, remarkable lattices, and cryptography. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 643–673. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_23
25. Feneuil, T., Joux, A., Rivain, M.: Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 541–572. Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_19
26. Feneuil, T., Joux, A., Rivain, M.: Shared permutation for syndrome decoding: new zero-knowledge protocol and code-based signature. Des. Codes Cryptogr. **91**(2), 563–608 (2023). https://doi.org/10.1007/s10623-022-01116-1
27. Feneuil, T., Maire, J., Rivain, M., Vergnaud, D.: Zero-knowledge protocols for the subset sum problem from MPC-in-the-head with rejection. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 371–402. Springer, Heidelberg (Dec 2022). https://doi.org/10.1007/978-3-031-22966-4_13
28. Feneuil, T., Rivain, M.: Threshold linear secret sharing to the rescue of MPC-in-the-head. Cryptology ePrint Archive, Report 2022/1407 (2022), `https://eprint.iacr.org/2022/1407`
29. Feo, L.D.: Mathematics of isogeny based cryptography (2017)
30. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM **33**(4), 792–807 (Oct 1986)
31. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987). https://doi.org/10.1145/28395.28420
32. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. Journal of the ACM **38**(3), 691–729 (1991)
33. Grochow, J.A., Qiao, Y.: On the complexity of isomorphism problems for tensors, groups, and polynomials I: Tensor isomorphism-completeness. In: Lee, J.R. (ed.) ITCS 2021. vol. 185, pp. 31:1–31:19. LIPIcs (Jan 2021). https://doi.org/10.4230/LIPIcs.ITCS.2021.31
34. Haviv, I., Regev, O.: On the lattice isomorphism problem. CoRR **abs/1311.0366** (2013), `http://arxiv.org/abs/1311.0366`
35. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 21–30. ACM Press (Jun 2007). https://doi.org/10.1145/1250790.1250794
36. Ji, Z., Qiao, Y., Song, F., Yun, A.: General linear group action on tensors: A candidate for post-quantum cryptography. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part I. LNCS, vol. 11891, pp. 251–281. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-36030-6_11
37. Joux, A.: Various approaches to signatures schemes. Invited talk at CBCrypto'23 (April 2023)
38. Kiayias, A., Papadopoulos, S., Triandopoulos, N., Zacharias, T.: Delegatable pseudorandom functions and applications. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 669–684. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516668
39. Lang, S.: Algebra. Springer, New York, NY (2002). https://doi.org/https://doi.org/10.1007/978-1-4613-0041-0
40. Maurer, U.M.: Abstract models of computation in cryptography. In: Smart, N.P. (ed.) Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3796, pp. 1–12. Springer (2005). https://doi.org/10.1007/11586821_1
41. Melchor, C.A., Gama, N., Howe, J., Hülsing, A., Joseph, D., Yue, D.: The return of the sdith. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 564–596. Springer (2023). https://doi.org/10.1007/978-3-031-30589-4_20, `https://doi.org/10.1007/978-3-031-30589-4\_20`

42. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (May 1996). https://doi.org/10.1007/3-540-68339-9_4
43. Sendrier, N., Simos, D.E.: The hardness of code equivalence over and its application to code-based cryptography. In: Gaborit, P. (ed.) Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013. pp. 203–216. Springer, Heidelberg (Jun 2013). https://doi.org/10.1007/978-3-642-38616-9_14
44. Tang, G., Duong, D.H., Joux, A., Plantard, T., Qiao, Y., Susilo, W.: Practical post-quantum signature schemes from isomorphism problems of trilinear forms. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 582–612. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_21