Zero-Knowledge Proofs from the Action Subgraph

Giacomo Borin, Edoardo Persichetti and Paolo Santini

University of Trento, Sapienza University of Rome, Florida Atlantic University and Marche Polytechnic University

Abstract

In this work, we describe a technique to amplify the soundness of zero-knowledge proofs of knowledge for cryptographic group actions.

1 Introduction

The recent call for signatures issued by the National Institute of Standards and Technology (NIST) [15] has rekindled the community's interest in trying to produce efficient post-quantum schemes from a variety of assumptions. Indeed, NIST is explicitly looking for signature schemes that are not based on lattices; this has led to an acceleration in the development of schemes from other areas such as coding theory, multivariate equations, and isogenies. In terms of the former, in particular, the last few years have seen the signature landscape evolve quite remarkably: considering that no viable code-based signature schemes were proposed for NIST's original call in 2017, we now instead have, to cite but a few, hash-and-sign schemes [9], protocols based on code equivalence [7, 3], MPC-in-the-head [13, 10, 11] and even rank metric [2, 8]. Among these, the protocols exploiting the MPC-in-the-head framework certainly represent a very promising approach, especially thanks to recent developments [1, 12] which have a very positive impact on its performance.

1.1 Our Contribution

In this work, we explore the possibility of employing the MPC-in-the-head techniques on top of group actions, with only minimal assumptions; most importantly, we do not require the commutativity property, which is often the biggest obstacle for protocol design. Indeed, the only object that can be formulated as a commutative group action appears in the context of isogenies, with the work of CSI-FiSh [6], although this is only practical under very specific circumstances. We show that it is possible to translate the framework successfully to this case, although it requires some non-trivial modifications to enable the secret sharing which is at its core. We therefore provide a fully general protocol that yields a proof of knowledge for cryptographic group actions. We then investigate the

specific case of code-based group actions, both in the Hamming metric, and the rank metric, showing how it is possible to design a signature scheme by applying the Fiat-Shamir transformation. We compare the performance profile obtained in this way with the respective schemes from literature [3, 8] and highlight the advantages and disadvantages of our technique.

2 Notation and Background

In this section we establish the notation that we will use throughout the paper, as well as recall basic concepts about linear codes and sigma protocols.

2.1 Notation

As usual, we use $\mathbb{F}_q$ to indicate the finite field with q elements and $\mathbb{F}_q^*$ to indicate its multiplicative group. Throughout the paper we will use denote with capital letters object such as sets and groups, and with lowercase letters their elements. We will use instead boldface letters to denote vectors and matrices. Given a matrix $\mathbf{A}$ over $\mathbb{F}_q$, we write $\mathbf{a}_i$ to indicate its i-th column. The general linear group formed by the non-singular $k \times k$ matrices over $\mathbb{F}_q$ is indicated as GL_k. For an ordered set J, we write $\mathbf{A}_J$ to indicate the matrix formed by the columns of $\mathbf{A}$ that are indexed by the elements in J; equivalent notation is adopted for vectors. The identity with size k is indicated as $\mathbf{I}_k$, while $\mathbf{0}$ denotes the null-matrix (its dimensions will always be clear from the context). We denote by S_n the symmetric group on n elements, and consider its elements as permutations of n objects. We represent permutations as n-tuples of the form $\pi := \{i_1, i_2, \cdots, i_n\}$, so that for $j = 1, \cdots, n$, it holds that $\pi(j) = i_j$. Given a matrix $\mathbf{A}$, we write $\pi(\mathbf{A})$ to indicate the matrix resulting from the action of π on the columns of $\mathbf{A}$. We denote by M_n the set of monomial tranformations, that is, transformations of the form $\mu := (\pi, \mathbf{v})$ with $\pi \in S_n$ and $\mathbf{v} \in \mathbb{F}_q^{*n}$, acting as follows

$$\mu(\mathbf{A}) = \mu\big((\mathbf{a}_1, \cdots, \mathbf{a}_n)\big) = (v_1 \mathbf{a}_{\pi^{-1}(1)}, v_2 \mathbf{a}_{\pi^{-1}(2)}, \cdots, v_n \mathbf{a}_{\pi^{-1}(n)}).$$

2.2 Cryptographic Group Actions

A *group action* is a well-known object in mathematics. It can be described as a function, as shown below, where X is a set and G a group.

$$\begin{aligned} \star : G \times X &\to X \\ (g, x) &\to g \star x \end{aligned}$$

A group action's only requirement is to be *compatible* with the group; using multiplicative notation for G, and denoting with e its identity element, this means that for all $x \in X$ we have $e \star x = x$ and that moreover for all $g, h \in G$, it holds that $h \star (g \star x) = (h \cdot g) \star x$. A group action is also said to be:

- *Transitive*, if for every $x, y \in X$, there exists $g \in G$ such that $y = g \star x$;
- *Faithful*, if there does not exist a $g \in G$ such that $x = g \star x$ for all $x \in X$, other than the identity;
- *Free*, if an element $g \in G$ is equal to identity whenever there exists an $x \in X$ such that $x = g \star x$;
- *Regular*, if it is free and transitive.

The adjective *cryptographic* is added to indicate that the group action in question has additional properties that are relevant to cryptography. For instance, a cryptographic group action should be *one-way*, i.e. given randomly chosen $x, y \in X$, it should be hard to find $g \in G$ such that $g \star x = y$ (if such a g exists). Indeed, the problem of finding this element is known as the *vectorization* problem, or sometimes *Group Action Inverse Problem (GAIP)*.

Problem 1 (GAIP)**.** *Given x and y in X, find, if any, an element $g \in G$ such that $y = g \star x$.*

A related problem asks to compute the action of the product of two group elements, given the result of the individual actions on a fixed element. This is known as the *parallelization* problem, and it corresponds to, essentially, the computational version of the Diffie-Hellman problem, formulated for generic group actions. A definition is given next.

Problem 2 (cGADH)**.** *Given x, $g \star x$ and $h \star x$, for $g, h \in G$, compute $(g \cdot h) \star x$.*

In fact, the analogy to the case of discrete logarithms is easily drawn, once one realizes that this is simply the group action given by the exponentiation map on finite cyclic groups.

Finally, other useful properties for group actions include those that make it *effective*, such as for instance the existence of efficient (probabilistic polynomial-time) algorithms for membership testing, sampling, computation (of the group operation and $\star$) etc.

2.3 Coding Theory

A linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ is a k-dimensional subspace of $\mathbb{F}_q^n$. The quantity $R = k/n$ is called code rate, and any vector $\mathbf{c} \in \mathcal{C}$ is called codeword. A canonical representation for a code is through a generator matrix, that is, a rank-k matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ such that $\mathcal{C} = \{\mathbf{uG} \mid \mathbf{u} \in \mathbb{F}_q^k\}$. Any code admits multiple generator matrices: for any $\mathbf{S} \in \mathrm{GL}_k$, which can be thought of as a change of basis, it holds that $\mathbf{SG}$ and $\mathbf{G}$ generate the same code. The dual code $\mathcal{C}^\perp$ is the set of all vectors that are orthogonal to codewords in $\mathcal{C}$, that is, $\mathcal{C}^\perp = \{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{cv}^\top = 0,\ \forall \mathbf{c} \in \mathcal{C}\}$. It is easy to see that $\mathcal{C}^\perp$ is linear subspace of $\mathbb{F}_q^n$ with dimension $r = n - k$ (which is normally called redundancy). The dual code is generated by a rank-r matrix $\mathbf{H} \in \mathbb{F}_q^{r \times n}$, which is called parity-check matrix and is such that $\mathbf{GH}^\top = \mathbf{0}$. Obviously, for any $\mathbf{S} \in \mathrm{GL}_r$, $\mathbf{H}$ and $\mathbf{SH}$ are parity-check matrices for the same code.

For $J \subseteq \{1, \cdots, n\}$, we write $\mathcal{C}_J := \{\mathbf{c}_J \mid \mathbf{c} \in \mathcal{C}\}$. We say that a set J with size k is an *information set* for a code $\mathcal{C}$ if, for any two distinct $\mathbf{c}, \mathbf{c}' \in \mathcal{C}$, it holds that $\mathbf{c}_J \neq \mathbf{c}'_J$, which implies that $\mathcal{C}_J$ contains q^k elements. Equivalently, J is an information set if, for $\mathbf{G}$ being a generator matrix for $\mathcal{C}$, it holds that $\mathbf{G}_J$ is non-singular. We say that a generator matrix $\mathbf{G}$ is in *systematic form* if $\mathbf{G} = \big(\mathbf{I}_k \, \mathbf{V}\big)$, for $\mathbf{V} \in \mathbb{F}_{k \times n}$. This matrix exists whenever $J = \{1, \cdots, k\}$ is an information set: starting from any generator matrix $\mathbf{G}$, we obtain the one in systematic form as $\mathbf{G}_J^{-1}\mathbf{G}$. Also, the systematic matrix is an invariant under changes of basis: if $\mathbf{G}' = \mathbf{S}\mathbf{G}$, then its systematic form is $\mathbf{G}'^{-1}_J \mathbf{G}' = \mathbf{G}_J^{-1}\mathbf{S}^{-1}\mathbf{S}\mathbf{G} = \mathbf{G}_J^{-1}\mathbf{G}$.

Normally, linear codes are measured using the Hamming metric. This metric assigns a *weight* to each codeword by simply counting the number of non-zero positions in it, i.e. $\mathsf{wt}_\mathsf{H}(\mathbf{c}) = \mid \{c_i : c_i \neq 0\} \mid$. The *distance* between two words is then defined as the number of positions in which they differ, i.e. $d_H(\mathbf{c}, \mathbf{c}') = \mathsf{wt}_\mathsf{H}(\mathbf{c} - \mathbf{c}')$. In this case, *isometries*, i.e. map which preserve the weight, are given by permutations, in the simplest case; these can be generalized to the so-called monomial maps, via some non-zero scaling factors. To be precise, we represent permutations as n-tuples of the form $\pi := \{i_1, i_2, \cdots, i_n\}$, so that for $\mathbf{c} = (c_1, \cdots, c_n)$, it holds that

$$\pi(\mathbf{c}) = \big(c_{i_1}, \cdots, c_{i_n}\big).$$

We then denote by M_n the set of monomial transformations, that is, transformations of the form $\mu := (\pi, \mathbf{v})$ with $\pi \in S_n$ and $\mathbf{v} \in \mathbb{F}_q^{*n}$, acting as follows

$$\mu(\mathbf{c}) = \pi(\mathbf{c}) \begin{pmatrix} v_1 & & & \\ & v_2 & & \\ & & \ddots & \\ & & & v_n \end{pmatrix}.$$

The notion of linear codes can be generalized to the case where each codeword is a matrix, instead of a vector; more precisely, $m \times n$ matrices over $\mathbb{F}_q$. We talk then about $[m \times n, k]$ *matrix code*, which can be seen as a k-dimensional subspace $\mathcal{C}$ of $\mathbb{F}_q^{m \times n}$. These objects are usually measured with a different metric, known as *rank* metric, where the weight of each codeword corresponds to its rank (as a matrix), that is $\mathsf{wt}_\mathsf{R}(\mathbf{C}) = \mathsf{Rank}(\mathbf{C})$, and the distance between two matrices $\mathbf{C}, \mathbf{C}'$ is defined as $d_R(\mathbf{C}, \mathbf{C}') = \mathsf{wt}_\mathsf{R}(\mathbf{C} - \mathbf{C}')$. In this case, isometries are maps which preserve the rank of a matrix, and are thus identified by two non-singular matrices $\mathbf{A} \in \mathrm{GL}_m$ and $\mathbf{B} \in \mathrm{GL}_n$ acting respectively on the left and on the right of each codeword, by multiplication.

In both of the metrics defined above, then, we can formulate a notion of *equivalence* in the same way, by calling equivalent two codes which are connected by an isometry. In other words, two linear codes $\mathcal{C}$ and $\mathcal{C}'$ are *linearly equivalent* if $\mathcal{C}' = \mu(\mathcal{C})$, and two matrix codes $\mathcal{C}$ and $\mathcal{C}'$ are *matrix equivalent* if $\mathcal{C}' = \mathbf{A}\mathcal{C}\mathbf{B}$. Note that the notion of *permutation equivalence* is just a special case of linear equivalence (corresponding to a trivial scaling vector $\mathbf{v}$), yet is often treated separately for a variety of reasons of both historical and practical nature (for instance, certain solvers behave quite differently).

2.4 Code Equivalence Problems

The problem of determining whether two codes are equivalent, in the respective senses described above, arises quite naturally in coding theory. We report below the respective formulations.

Problem 3. Permutation Equivalence (PEP)
Given*: two k-dimensional linear codes $\mathcal{C}, \mathcal{C}' \subseteq \mathbb{F}_q^n$*
Goal*: Determine if there exists $\pi \in S_n$ such that $\mathcal{C}' = \pi(\mathcal{C})$.*

Problem 4. Linear Equivalence (LEP)
Given*: two k-dimensional linear codes $\mathcal{C}, \mathcal{C}' \subseteq \mathbb{F}_q^n$*
Goal*: Determine if there exists $\mu \in M_n$ such that $\mathcal{C}' = \mu(\mathcal{C})$.*

Problem 5. Matrix Code Equivalence (MCE)
Given: *Two k-dimensional matrix codes $\mathcal{C}, \mathcal{C}'$.*
Goal: *Determine if there exist $\mathbf{A} \in \mathrm{GL}_m, \mathbf{B} \in \mathrm{GL}_n$ such that $\mathcal{C}' = \mathbf{A}\mathcal{C}\mathbf{B}$.*

Note that, as above, we have stated PEP as a separate problem, even though this is just a special case of LEP. Also, all of the above problems are formulated as decisional problem, as is traditional, but in cryptographic applications we are most often interested in their computational version. Rather extensive treatments of their hardness is given, for instance, in [5, 8].

3 Proving Equivalence via Random Paths

We now describe our new proof-of-knowledge strategy for cryptographic group actions. As we mentioned in the preamble, we formulate our protocols considering the most general possible setting for group actions. Thus, the only properties we require for our construction are that the action is transitive and faithful.

3.1 The Action Graph

Let $\mathcal{O}(x)$ denote the orbit of x under the action of G, that is

$$\mathcal{O}(x) = \{g \star x \mid x \in G\}.$$

If the action of G is faithful, then $\mathcal{O}(x)$ contains $|G|$ elements. Also, one of them is the public key element $x' = g \star x$ for some secret $g \in G$. We can represent such an orbit through a simple, ordered graph $\mathcal{G}$ whose vertices are the elements in $\mathcal{O}(x)$. Any pair of vertices (x_1, x_2) is connected by a pair of edges (g_1, g_2) such that $x_2 = g_1 \star x_1$ and $x_1 = g_2 \star x_1$ when $\star$ is faithful, $g_2 = g_1^{-1}$. We call $\mathcal{G}$ the *action graph* of G on $X = \mathcal{O}(x)$. We observe also that $\mathcal{G}$ is connected: any pair of vertices (x_1, x_2) is connected by two edges (one for each direction).

Let us now assume that the prover performs a random walk on $\mathcal{G}$, starting from x and ending in x_N. This can be done by generating N random elements of

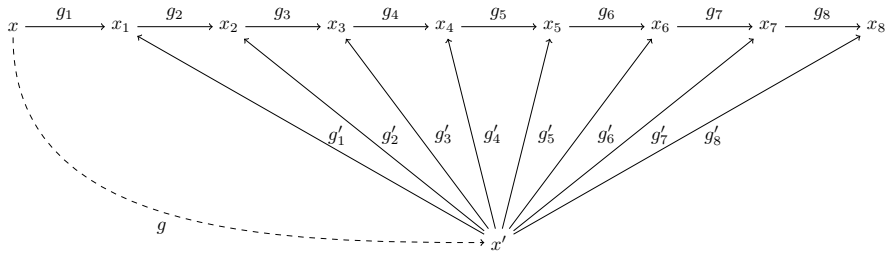

Figure 1: Example of action subgraph, for $N = 8$.

G uniformly at random $\{g_1, g_2, \cdots, g_N\}$, and letting them act on x sequentially, that is

$$x_0 = x \xrightarrow{g_1} x_1 \xrightarrow{g_2} x_2 \xrightarrow{g_3} \cdots \xrightarrow{g_{N-1}} x_{N-1} \xrightarrow{g_N} x_N.$$

Notice that, for $i \geqslant 1$, it holds that

$$x_i = (g_i \cdot g_{i-1} \cdots g_2 \cdot g_1) \star x.$$

From now on, we will refer to the path going from x to x_N as *random path*. We can think that the random path defines a subgraph $\mathcal{S} \subset \mathcal{G}$, built as follows:

- it has $N + 2$ vertices $\{x_0 = x, x_1, \cdots, x_N, x'\}$;
- it has N edges of the form (x_{i-1}, x_i), for $i = 1, \cdots, N$. Each such edge is labelled with g_i;
- it has N edges (x', x_i), for $i = 1, \cdots, N$. Each such edge is labelled with $g'_i = g_i \cdot g_{i-1} \cdots g_1 \cdot g^{-1}$.

We will refer to $\mathcal{S}$ as *action subgraph*; an example of how this graph looks like is shown in Figure 1. Notice that the graph may be enriched with additional edges: indeed, the vertices in $\mathcal{S}$ would form a connected graph, assuming one draws all edges. However, as we describe in the following, the edges we are considering we are considering are the only ones which can be used to build a zero knowledge proof system, and (some) missing edges can be easily recomputed from the information which is provided by the prover.

We now observe that, when the random path $x \mapsto x_N$ has been genuinely generated (i.e., using elements $g_1, \cdots, g_N$ generated uniformly at random), then knowing a path $x \mapsto x'$ is possible only if one knows also a secret element g such that $g \star x = x'$. Indeed, let us consider a path $x \mapsto x'$ in $\mathcal{S}$ along the edges $P = \{(x_0, x_1), (x_1, x_2), \cdots, (x_{i-1}, x_i), (x_i, x')\}$, for some $i \in \{1, \cdots, N\}$. Since (x_i, x') is labeled by $g'_i = g_i \cdots g_1 \cdot g^{-1}$ and each (x_{j-1}, x_j) has label g_j, then the secret g can be easily recovered. Indeed, consider that

$$\begin{aligned}(g'_i)^{-1} \cdot (g_1 \cdots g_i) &= \left((g_i \cdots g_1) \cdot g^{-1}\right)^{-1} \cdot (g_1 \cdots g_i) \\ &= \left(g \cdot (g_i \cdots g_1)^{-1}\right) \cdot (g_1 \cdots g_i) \\ &= g. \end{aligned} \tag{1}$$

3.2 Proofs of Knowledge from the Action Subgraph

In this section we describe how to build a ZK proof of knowledge from the considerations in the previous section. Basically, the idea is that of proving knowledge of a path $x \mapsto x'$ without revealing all the edges. This would prevent an adversary from recovering the secret key by applying (1). The subgraph will be constructed in the usual ZK fashion, i.e., elements $g_1, \cdots, g_N$ will be sampled uniformly at random from G and the prover will commit to the corresponding random path, i.e., to everything that binds all the edges from x to x_N. Then, the verifier will select a random *interruption*, that is, an index $i \in \{1, \cdots, N\}$. The prover has to show that the graph has been honestly obtained, i.e., it is the same regardless of the interruption (the challenge value). To do this, he will provide:

- all the edges from x to x_i: this can be done by revealing g_j, for $j \leqslant i$;
- the path from x' to x_{i+1}: this can be done by revealing g';
- all the edges from x_{i+1} to x_N: this can be done by revealing g_j, for $i+1 \leqslant j \leqslant N$.

A proof of knowledge for the above paradigm would be then be of the form $P(i) = \left(g'_i, \{g_j\}_{j \neq i+1}\right)$. If $i = N$ (i.e., no interruption is actually asked) the prover instead reveals all labels $\{g_j\}_{1 \leqslant j \leqslant N}$. To sum up, the proof of knowledge is a function of the sole challenge index $i \in \{1, \cdots, N\}$, and has the following form

$$P(i) = \begin{cases} \left(g'_i, \{g_j\}_{j \neq i+1}\right) & \text{if } i < N, \\ \{g_j\}_{1 \leqslant j \leqslant N} & \text{if } i = N. \end{cases} \tag{2}$$

An example of how the proof of knowledge is constructed is shown in Figure 2.

Observe that, from the information provided in the proof as in (2), one can also recover all the edges g'_j with $j > i+1$, but this is not useful to retrieve the secret g. Indeed, the vertices in the subgraph can be divided into groups: $V = \{x, x_1, \cdots, x_i\} \subset \mathcal{O}(x)$ and $V' = \{x_{i+1}, x_{i+1}, \cdots, x_N\} \subset \mathcal{O}(x')$. What the verifier does is checking that both V and v' are connected subgraphs. However, the prover never reveals an edge connecting a vertex in V with one in V': this guarantees that g cannot be recovered from the proof $P(i)$. A representation of the situation is depicted in Figure 3.

The corresponding ZK-ID protocol is shown in Figure 4. Notice that all group elements $g_1, \cdots, g_i$ can be compactly communicated using generating seeds, while for g'_i this is not possible. When $i = N$ (i.e., no interruption is asked), the proof is obtained by revealing all seeds and each edge in the random path is verified. Using the standard technique of using a PRG tree, revealing the random edges (i.e., edges labeled by some g_j) costs $\lambda \log_2(N)$ bits in case $i \neq N$, and only λ bits in case $i = N$.

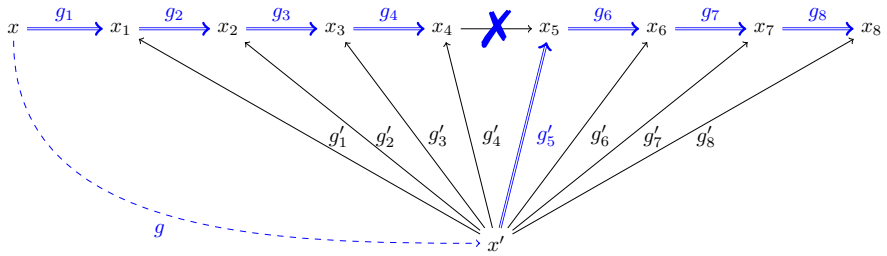

Figure 2: Example of proof of knowledge constructed on the action subgraph, for the interruption $i = 4$. The edges that form the path from x to x' are highlighted in blue; the edges that are revealed by the prover are highlighted with double arrows.

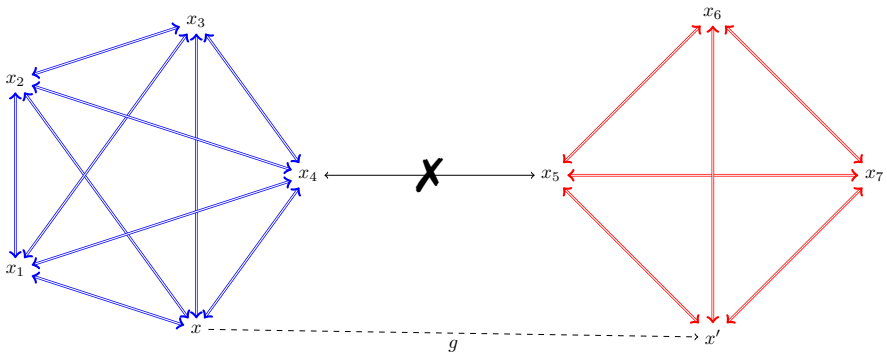

Figure 3: Example of all edges that one learns in case the interruption is $i = 4$.

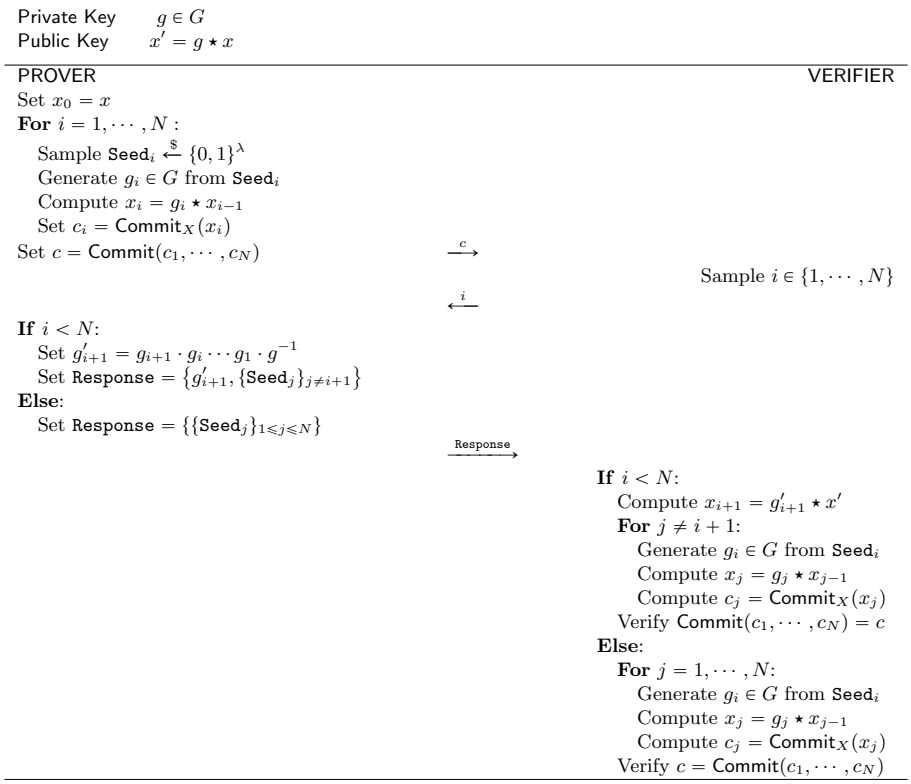

Figure 4: A single round of the new ZK-ID protocol based on interruptions in the action subgraph

Notice that we generically consider two commitment functions. While Commit can be any generic commitment function, Commit_X is specific to how elements of X are represented and (as we detail when we focus on the code equivalence group action). We generically assume that Commit_X requires a larger computational cost, with respect to Commit. As we argue in the following, our protocol allows to trade the signature size with the verification time: with some increase in the signature size, one can greatly reduce the number of times the function Commit_X is computed, on the verifier's side. This fact will become very important when focusing on CEP, since the function Commit_X will require to compute systematic forms.

We now study the security properties of the resulting protocol. Seeing that it is complete is trivial, so we focus on Zero Knowledge and soundness.

Zero Knowledge We want to show that a simulator that knows, in advance, the challenge value can produce a valid transcript which is statistically distributed as the one that would be produced by an honest prover. To this end,

consider a simulator that, on input $i \in \{1, \cdots, N\}$, proceeds as follows:

- if $i = N$, it samples seeds $\mathtt{Seed}_1, \cdots, \mathtt{Seed}_N$ and generates the random path and the commitments; then, outputs all the seeds. Obviously, this output is statistically distributed as the one of the honest prover;

- if $i < N$, it selects random seeds $\mathtt{Seed}_1, \cdots, \mathtt{Seed}_i$ and uses them to compute $\{g_j\}_{j \leqslant i}$ and $\{x_j\}_{j \leqslant i}$ inductively as $x_j = g_j \star x_{j-1}$. Then, the simulator selects uniformly at random $\widetilde{g} \in G$ and uses it to compute $\widetilde{x}_{i+1} = \widetilde{g} \star x'$. Finally, if $i + 1 < N$, it selects the remaining random seeds $\mathtt{Seed}_{i+2}, \cdots, \mathtt{Seed}_N$, it uses them to compute $\{g_j\}_{j \geqslant i+2}$ and to inductively compute $\widetilde{x}_{i+2} = g_{i+2} \star \widetilde{x}_{i+1}, \ldots, \widetilde{x}_N = g_N \star \widetilde{x}_{N-1}$. The simulator hashes $x_1, \ldots, x_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_N$ to produce the commitments, which indistinguishable from those of a honest execution. As output, the simulator provides $\mathtt{Response} = \{\widetilde{g}, \{\mathtt{Seed}_j\}_{j \neq i+1}\}$. The elements $\widetilde{g}$ and $\{\mathtt{Seed}_j\}_{j \neq i+1}$ are independent and uniformly distributed over G. This is also the case of the response of an honest prover because g'_{i+1} has the form

$$g'_{i+1} = \underbrace{g_{i+1}}_{\substack{\text{Secret and} \\ \text{ephemeral}}} \cdot \underbrace{g_i \cdot g_{i-1} \cdots g_1}_{\substack{\text{Public and} \\ \text{ephemeral}}} \cdot \underbrace{g^{-1}}_{\text{Secret}}.$$

  Since g_{i+1} is independent from the $\{g_j\}_{j \leqslant i+1}$, g'_{i+1} is also distributed uniformly at random independently from $\{\mathtt{Seed}_j\}_{j \neq i+1}\}$. One can immediately check that the simulator's response leads to a valid transcript.

Special Soundness Let us consider two accepting transcripts with same commitment, but different challenge values, say, i_0, i_1. Without loss of generality, let $i_0 < i_1$. We show that there is an efficient and rather easy extractor that can compute a witness, on input such two transcripts. The idea of the proof is that of showing that, using the edges contained in the two transcripts, the extractor is always able to obtain a path $x \mapsto x'$. Continuing with the graph analogy, either hash collisions are found or indeed the two following paths in the action subgraph have the same ending vertex

$$x \mapsto x_{i_0+1}, \quad x' \mapsto x_{i_0+1}.$$

So, the extractor knows a path $x \mapsto x'$, which corresponds to the witness.

Remark 1. *The protocol in Figure 4 can be modified, to achieve trade-offs between signature size and number of group actions computations. Indeed, verification of the commitments for the intermediate vertices in the path $x_{i+1} \mapsto x_N$ is not necessary. In such a case, the prover may instead construct a Merkle tree from $c_1, \cdots, c_N$, and provide the Merkle proofs for such vertices. This would reduce the number of times Commit_X is computed, on the verifier's side, but increases the signature size since Merkle proofs are now needed.*

Since we proved that the protocol achieves special soundness, it has knowledge error given by

$$\varepsilon = \frac{1}{N}.$$

3.3 Using multiple key pairs and unbalanced challenges

As in [3], the soundness of the protocol can be amplified by using more secrets to create the public values. Namely, the prover can choose M secrets $g^{(1)}, \cdots, g^{(M)}$ and compute the corresponding public values as $x'_i = g^{(i)} \star x$, for $i = 1, 2, \cdots, M$. The action subgraph is computed in the very same way, but now the verifier challenges the prover by specifying, together with the position of the interruption, also the public value from which the proof should be provided. This way, the soundness error is reduced from N to $\varepsilon = MN$.

Also, when considering the Fiat-Shamir transformation of parallel repetitions, one can use fixed weight challenges. In such a case, the challenge vector would become a length-t vector (assuming t parallel repetitions are executed). In $t - w$ rounds the verifier chooses $i = N$ (i.e., no interruption), while in the other rounds he selects the interruption and the public value. The resulting protocol would keep the special soundness property, and would be characterized by an overall soundness error of

$$\varepsilon = \binom{t}{w} \Big(M(N-1)\Big)^w.$$

For the $t - w$ rounds with $i = N$, the prover is asked only for the seeds, which can be compactly represented by using a unique tree PRNG for all rounds (i.e., with tN leaves in the base layer).

4 The Case of Code Equivalence Group Actions

As a concrete instantiation of the previous protocol, we consider the code equivalence group action. To this end, we set $X := \mathscr{O}(\mathcal{C})$. The secret group element is $g := \mu \in S_n$, while the public key is $\mathcal{C}' = \mu(\mathcal{C}) \in \mathscr{O}(\mathcal{C})$. The commitment function Commit_X is implemented as the hash of the systematic generator matrix, while Commit can be any cryptographically secure hash function. Given the interest in digital signatures, we already consider the scheme resulting by application of the Fiat-Shamir transform on t parallel executions of the paradigm presented in Figure 4. We consider that M public keys are employed, and that an interruption is asked in only w rounds. The resulting scheme is shown in Figure 5.

In the description of the scheme, we have implicitly defined standard functions such as $\mathsf{SeedTree}$, which implement the PRNG tree, and Path which is used to compute the intermediate seeds in a PRNG tree. We have also used a function Commit which is used to generate the challenge vector $\left(\mathtt{Ch}^{(1)}, \cdots, \mathtt{Ch}^{(t)}\right)$

Private Key $\quad \mu_1, \cdots, \mu_M \in M_n$
Public Key $\quad$ linear codes $\mathcal{C}'_i = \mu_i(\mathcal{C})$, for $i = 1, \cdots, M$

PROVER	VERIFIER
Sample $\mathtt{Salt}, \mathtt{MSeed} \xleftarrow{\$} \{0;1\}^\lambda$	
Generate $\mathtt{Seed}^{(1)}, \cdots, \mathtt{Seed}^{(t)} = \mathsf{SeedTree}(\mathtt{Mseed})$	
Set $\{\mathcal{C}_0^{(j)}\}_{1 \leqslant j \leqslant t} = \mathcal{C}$	
For $j = 1, \cdots, t$:	
Generate $\mathtt{Seed}_1^{(j)}, \cdots, \mathtt{Seed}_N^{(j)} = \mathtt{SeedTree}(\mathtt{MSeed}^{(j)})$	
For $i = 1, \cdots, N$:	
Generate $\tau_i^{(j)} = \mathsf{PRNG}(\mathtt{Seed}_i^{(j)})$	
Compute $\mathcal{C}_i^{(j)} = \sigma_i^{(j)}(\mathcal{C}_{i-1}^{(j)})$	
Compute $c_i^{(j)} = \mathsf{Commit}_X\big(\mathcal{C}_i^{(j)}, \mathtt{Salt}\big)$	
Compute $c = \mathsf{Commit}\Big(\{c_i^{(j)}\}_{\substack{1 \leqslant j \leqslant t \\ 1 \leqslant i \leqslant N}}\Big)$	
Set $\big(\mathtt{Ch}^{(1)}, \cdots, \mathtt{Ch}^{(t)}\big) = \mathsf{Challenge}(c, \mathtt{Salt}, m)$	
Set $J = \{j \mid \mathtt{Ch}^{(j)} = N\}$	
Compute $\mathtt{MSeedPath} = \mathsf{Path}(\mathtt{Mseed}, J)$	
For j such that $\mathtt{Ch}^{(j)} \neq N$:	
Parse $\mathtt{Ch}^{(j)}$ as $\big(i, z_j\big)$	
Set $\mu'^{(j)}_{i+1} = \tau^{(j)}_{i+1} \cdot \tau^{(j)}_i \cdots \tau^{(j)}_1 \cdot \mu^{-1}_{z_j}$	
Set $\mathtt{SeedPath}^{(j)} = \mathsf{Path}\big(\mathtt{MSeed}^{(j)}, i\big)$	
	$\xrightarrow{\mathtt{Salt}, c, \left\{\mu'^{(j)}_{z_j}, \mathtt{SeedPath}^{(j)}\right\}_{j \notin J}}$

Figure 5: The LESS scheme based on the action subgraph

for all rounds. Exactly $t - w$ out of these t values are equal to N, as these are executions in which no interruption is asked. The remaining values are in the form $\big(i, z_j\big)$: i is the index of the interruption, while z_j is the index of the public code from which one should restart.

We have not reported details for the verification procedure, as it basically follows the description we provided in the previous section. The verifier is able to locally recompute all codes $\mathcal{C}_i^{(j)}$, with $1 \leqslant i \leqslant N$ and $1 \leqslant j \leqslant t$. For the rounds where the challenge is N, the verifier generates all the random monomials $\tau_i^{(j)}$. Instead, in each of the other rounds, he will use also one of the edges connecting one of the public key codes to a vertex in the action subgraph.

Signature size and instances Let $\ell_\mu = n \log_2(n) + n \log_2(q-1)$ denote the binary size of a monomial transformation. Then, the size of the resulting signature, in bits, is given by the following formula:

$$\underbrace{3\lambda}_{\mathtt{Salt},\ c} + \underbrace{w\lambda \log_2\left(\frac{t}{w}\right)}_{\mathtt{MSeedPath}} + w\Big(\underbrace{\ell_\mu}_{\mu'^{(j)}_{z_j}} + \underbrace{\lambda \log_2(N)}_{\mathtt{SeedPath}^{(j)}}\Big).$$

	M	Pk size	N	(t, w)	Sign size
PEP $(q, n, k) = (251, 235, 108)$	1	13.72	8	$(64, 25)$	7.67
			16	$(57, 20)$	6.51
			32	$(64, 16)$	5.60
	3	41.15	8	$(59, 18)$	5.64
			16	$(61, 15)$	5.02
			32	$(63, 13)$	4.62
LEP $(q, n, k) = (251, 198, 94)$	1	9.77	8	$(64, 25)$	11.69
			16	$(57, 20)$	9.73
			32	$(64, 16)$	8.18
	3	29.33	8	$(59, 18)$	8.53
			16	$(61, 15)$	7.43
			32	$(63, 13)$	6.71

Table 1: Comparison between LESS based on action subgraph and LESS-FM

Using PEP instead of LEP, the scheme would remain the same, with the only difference that now isometries are sampled from S_n. The expression for the signature size does not modify, but we need to replace ℓ_μ with $\ell_\pi = n \log_2(n)$, which is the number of bits required to represent a permutation.

Examples of resulting instances are shown in Table 1. Comparing with [4], we see that we are able to achieve competitive signature sizes with much smaller public keys. For instance, [4] reports a PEP instance with signatures of 5.25 kB and a 205.74 kB public key (for this instance, $M = 15$ public codes are used). Using the same code parameters, we can achieve a slightly larger signature using only one public key. We can also achieve smaller signatures, using only $M = 3$ public codes, resulting in a public key which is 41.15 kB (hence, 5 times smaller). Analogous gains hold for the LEP case. Namely, with respect to [4], we are able to achieve either smaller or comparable signatures, with the remarkable improvement that the public key sizes is drastically reduced.

Finally, we notice that one may obtain even more compact signatures by using larger values for N and/or larger values for t. This would come with a trade-off in the computational complexity: increasing these quantities lead to a larger number of generated codes (during signing and verification), hence to a larger number of systematic form computations.

Acknowledgments. During the write-up of this preprint, we became aware of another recent preprint [14] which verges on topics closely related to this work. Upon discussion with the author, we established that the two works are independent and concurrent, and agreed to keep them separate. We would like to thank the author of [14] for fruitful conversations on the subject.

References

[1] C. Aguilar-Melchor et al. "The return of the SDitH". In: *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*. Springer. 2023, pp. 564–596.

[2] N. Aragon et al. "Durandal: A Rank Metric Based Signature Scheme". In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Y. Ishai and V. Rijmen. Cham: Springer International Publishing, 2019, pp. 728–758.

[3] A. Barenghi et al. "LESS-FM: Fine-Tuning Signatures from the Code Equivalence Problem". In: *PQCrypto 2021*. Ed. by J. H. Cheon and J. Tillich. Vol. 12841. LNCS. Springer, 2021, pp. 23–43.

[4] A. Barenghi et al. "LESS-FM: fine-tuning signatures from the code equivalence problem". In: *Post-Quantum Cryptography: 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20–22, 2021, Proceedings 12*. Springer. 2021, pp. 23–43.

[5] A. Barenghi et al. "On the computational hardness of the code equivalence problem in cryptography". In: *Advances in Mathematics of Communications* 17.1 (2023), pp. 23–55.

[6] W. Beullens, T. Kleinjung, and F. Vercauteren. "CSI-FiSh: Efficient Isogeny based Signatures through Class Group Computations". In: *IACR Cryptol. ePrint Arch.* (2019), p. 498. URL: `https://eprint.iacr.org/2019/498`.

[7] J.-F. Biasse et al. "LESS is More: Code-Based Signatures Without Syndromes". In: *AFRICACRYPT*. Ed. by A. Nitaj and A. Youssef. Springer, 2020, pp. 45–65.

[8] T. Chou et al. "Take your MEDS: Digital Signatures from Matrix Code Equivalence". In: *AFRICACRYPT* (2023).

[9] T. Debris-Alazard, N. Sendrier, and J.-P. Tillich. "Wave: A new family of trapdoor one-way preimage sampleable functions based on codes". In: *ASIACRYPT*. Springer. 2019, pp. 21–51.

[10] T. Feneuil, A. Joux, and M. Rivain. "Shared permutation for syndrome decoding: New zero-knowledge protocol and code-based signature". In: *Designs, Codes and Cryptography* 91.2 (2023), pp. 563–608.

[11] T. Feneuil, A. Joux, and M. Rivain. "Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs". In: *Advances in Cryptology – CRYPTO 2022*. Vol. 13508. LNCS. Springer, 2022, pp. 541–572.

[12] T. Feneuil and M. Rivain. "Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head". In: *Cryptology ePrint Archive* (2022).

[13] S. Gueron, E. Persichetti, and P. Santini. "Designing a Practical Code-Based Signature Scheme from Zero-Knowledge Proofs with Trusted Setup". In: *Cryptography* 6.1 (2022), p. 5.

[14] A. Joux. *MPC in the head for isomorphisms and group actions*. Cryptology ePrint Archive, Paper 2023/664. https://eprint.iacr.org/2023/664. 2023. URL: https://eprint.iacr.org/2023/664.

[15] NIST. *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process*. URL: https://csrc.nist.gov/projects/pqc-dig-sig/standardization/call-for-proposals. 2023.