A Guide to the Design of Digital Signatures based on Cryptographic Group Actions

Giacomo Borin, Edoardo Persichetti, Federico Pintore,
Krijn Reijnders and Paolo Santini

IBM Research Zurich, Florida Atlantic University, University of Trento, Radboud University and Marche Polytechnic University

Abstract. Cryptography based on group actions has been studied since 1990. In recent years, however, the area has seen a revival, partially due to its role in post-quantum cryptography. For instance, several works have proposed signature schemes based on group actions, as well as a variety of techniques aimed at improving their performance and efficiency. Most of these techniques can be explained as transforming one Sigma protocol into another, while essentially preserving security. In this work, we present a unified taxonomy of such techniques. In particular, we describe all techniques in a single fashion, show how they impact the performance of the resulting protocols and analyse in detail how different techniques can be combined for optimal performance. Furthermore, to provide a tangible perspective, we apply the results of our analysis to the (group action-based) candidates in the current NIST call for digital signatures. This gives a full overview of the state of the art of signatures based on group actions, as well as a flexible tool which is easy to adapt and employ in the design of future schemes.

Keywords. Group Actions, Zero-Knowledge, Signature Scheme

1 Introduction

The recent standardization effort promoted by the National Institute of Standards and Technology (NIST) [1] has rekindled the community's interest in the design of efficient post-quantum cryptographic schemes from a variety of assumptions. In the latest call [35], NIST is explicitly looking for signature schemes that are not based on lattices, which has led to an acceleration in the development of schemes from other areas, such as coding theory, multivariate equations, isogenies and symmetric primitives.

Among these, cryptographic group actions represent a powerful tool. Formally introduced in 1990 by Brassard and Yung [15], cryptographic group actions have seen a renewed interest in recent years, thanks to their flexibility in constructing post-quantum signature schemes and other primitives based on various notions of isomorphism, be that isogenies between elliptic curves [16, 21], equivalence of linear or matrix codes [5, 9, 19], isomorphism of lattices [25], alternating

trilinear forms [38] and others. At the same time, another trend to build efficient signatures emerged, exploiting the MPC-in-the-head paradigm [33], leading to the development of efficient schemes, again, from a variety of mathematical assumptions [26, 27, 29], or even none, i.e. using only symmetric primitives [32].

Nowadays, the landscape of digital signatures based on group actions is broad. The different signature schemes that have been proposed so far are all built on a generic 3-pass identification protocol Π (more precisely, a Sigma protocol) which allow a prover to prove knowledge of a group element whose action sends a public element of the set on which the group acts into another public set element. The protocol Π has been instantiated from a variety of group actions, leading to the current wide range of group action-based digital signatures. Furthermore, several generic and instantiation-tailored techniques to increase the performance or achieve trade-offs between scheme parameters have been presented [5, 7, 9, 19, 22, 38], with the most recent proposals including also an adaptation of the MPC-in-the-head paradigm to the group action setting [31].

Thus, in practice, in the design of cryptographic primitives based on group actions, one has to consider a vast array of literature and variables to achieve optimal performance, as there is no unified description of these techniques and no clear insight into the performance impact of their combinations.

Our Contributions. This work aims to fill this gap in the literature, by presenting a complete taxonomy of the current techniques used in the design of signature schemes based on group actions. We do this in all generality, by using a unified description framework, and providing a detailed analysis of each technique's impact on performance and security. Table 1 summarises the effectiveness of all current techniques and fruitful combinations in terms of (maximum) signature size, public key size, signing time and verification time. Our work is organised as follows.

In Sections 3 and 4 we describe the individual optimisation techniques, dividing them between well-established ones (Section 3) and more recent and novel techniques (Section 4). Each of the discussed techniques will correspond to a column in Table 1. In particular:

- Section 3 details the Multiple Keys, Fixed-Weight Challenges and Seed Tree optimisations, corresponding to the columns in Table 1 labelled by **MK**, **FW** and **ST**, respectively;
- Section 4 describes the MPC-in-the-Head, Skipping Edges and Hypercube optimisations, corresponding to the columns in Table 1 labelled by **MPC**, **Skip** and **Cube**, respectively;
- Skipping Edges and Hypercube are specific for the MPC-in-the-Head setting and, to the best of our knowledge, they have not appeared in the literature before.

The rows of Table 1 report the results obtained for digital signatures constructed by applying the Fiat-Shamir transform [28] to the Sigma protocol Π, combined with various different choices of optimisation techniques. The efficiency of signature schemes is assessed by means of different quantifiers, each of them corresponding to a column in the table. In particular:

- the column labelled by **Max Sig Size** reports the maximum signature size where, to simplify reading, we have removed the "static" part consisting of salt and σ_2 (the challenge string) which is identical for all cases;
- the column **Pk Size** reports the size of the public key;
- the columns **Sign Time** and **Ver Time** report, respectively, the time required to perform signing and verification, expressed as a function of the system parameters.
- finally, the last column, labelled by **Condition**, describes the necessary condition on parameters to be satisfied for each particular combination. The first entry of each row provides the reference to the subsection of the paper where the efficiency of the corresponding signature scheme is analysed in full details.

The first row takes into consideration the signature from the basic protocol Π, which will thereafter serve as a reference, and comparison term. The next rows, then, consider the signatures obtained from Π when combined with a single optimisation: rows 2 to 4 consider one among Multiple Keys, Fixed-Weight Challenges and Seed Tree, while rows 5 to 8 focus on the MPC-in-the-Head optimisation, first by itself, and then combined with either Skipping Edges or Hypercube, which are "direct improvements" (i.e. can be applied exclusively in combination with MPC-in-the-Head).

In Section 5, we analyse more elaborate combinations of the optimisation techniques. The results obtained therein are summarised in the last 15 rows of Table 1. Furthermore, building on these, we show in Section 6 that several combinations of techniques are always superseded by other combinations, and can therefore be disregarded in most cryptographic designs. In particular, Theorems 2 and 3 show that, informally, Fixed-Weight Challenges almost always outperforms MPC-in-the-Head.

Lastly, based on our unified approach, we are able to effectively summarise the state of the current digital signatures schemes based on group actions: we do this in Section 7, where we also propose new parameters for some of these schemes, according to the guidelines emerging from Table 1, which are effectively an improvement over the original ones. This case of study shows how the comprehensive analysis provided in this work might represent an effective tool in designing and improving the efficiency of digital signatures based on group actions.

	Transformations						Bandwidth Cost (after Fiat-Shamir)		Comput. Cost		
Section	**MK**	**FW**	**ST**	**MPC**	**Skip**	**Cube**	**Max Sig Size** (excl. salt & σ_2)	**Pk Size**	**Sign Time**	**Ver Time**	**Condition**
Sec. 2.4	-	-	-	-	-	-	$t\ell_G$	$\ell_X + \lambda$	t	t	$t = \lambda$
Sec. 3.1	✓	-	-	-	-	-	$t\ell_G$	$s\ell_X + \lambda$	t	t	$(s+1)^t \geqslant 2^\lambda$
Sec. 3.2	-	✓	-	-	-	-	$(t-w)\lambda + w\ell_G$	$\ell_X + \lambda$	t	t	$\binom{t}{w} \geqslant \lambda$
Sec. 3.3	-	-	✓	-	-	-	$N_{\mathsf{seed}}\lambda + w\ell_G$	$\ell_X + \lambda$	t	t	$(s+1)^t \geqslant 2^\lambda$
Sec. 4.1	-	-	-	✓	-	-	$t((m-1)\lambda + \ell_G)$	$\ell_X + \lambda$	tm	tm	$(m+1)^t \geqslant 2^\lambda$
Sec. 4.2	-	-	-	✓	✓$_L$	-	$t\big((m-1)\lambda + \ell_G + 2\lambda\big)$	$\ell_X + \lambda$	tm	$t\left(1+\frac{m}{2}\right)$	$(m+1)^t \geqslant 2^\lambda$
Sec. 4.2	-	-	-	✓	✓$_R$	-	$t((m-1)\lambda + \ell_G)$	$\ell_X + \lambda$	tm	$t\left(1+\frac{m}{2}\right)$	$(m+1)^t \geqslant 2^\lambda$
Sec. 4.3	-	-	-	✓	-	✓	$t((m-1)\lambda + \ell_G)$	$\ell_X + \lambda$	$t\log(m)$*	$t\log(m)$	$(m+1)^t \geqslant 2^\lambda$
Sec. 5.1	-	✓	✓	-	-	-	$N_{\mathsf{seed}}\lambda + w\ell_G$	$\ell_X + \lambda$	t	t	$\binom{t}{w} \geqslant \lambda$
	✓	✓	✓	-	-	-	$N_{\mathsf{seed}}\lambda + w\ell_G$	$s\ell_X + \lambda$	t	t	$\binom{t}{w}s^w \geqslant \lambda$
Sec. 5.2	-	-	✓	✓	-	-	$t(\lceil\log(m)\rceil\lambda + \ell_G)$	$\ell_X + \lambda$	tm	tm	$(m+1)^t \geqslant 2^\lambda$
	✓	-	✓	✓	-	-	$t(\lceil\log(m)\rceil\lambda + \ell_G)$	$s\ell_X + \lambda$	tm	tm	$(sm+1)^t \geqslant 2^\lambda$
	-	✓	✓	✓	-	-	$N_{\mathsf{seed}}\lambda + w(\lceil\log(m)\rceil\lambda + \ell_G)$	$\ell_X + \lambda$	tm	tm	$\binom{t}{w}m^w \geqslant 2^\lambda$
	✓	✓	✓	✓	-	-	$N_{\mathsf{seed}}\lambda + w(\lceil\log(m)\rceil\lambda + \ell_G)$	$s\ell_X + \lambda$	tm	tm	$\binom{t}{w}(sm)^w \geqslant 2^\lambda$
	-	-	✓	✓	-	✓	$t(\log(m)\lambda + \ell_G)$	$\ell_X + \lambda$	$t\log(m)$*	$t\log(m)$	$(m+1)^t \geqslant 2^\lambda$
	✓	-	✓	✓	-	✓	$t(\log(m)\lambda + \ell_G)$	$s\ell_X + \lambda$	$t\log(m)$*	$t\log(m)$	$(sm+1)^t \geqslant 2^\lambda$
Sec. 5.3	-	-	✓	✓	✓$_L$	-	$t(\lceil\log(m)\rceil\lambda + \ell_G + 2\lambda)$	$\ell_X + \lambda$	tm	$t(1+\frac{m}{2})$	$(m+1)^t \geqslant 2^\lambda$
	✓	-	✓	✓	✓$_L$	-	$t(\lceil\log(m)\rceil\lambda + \ell_G + 2\lambda)$	$s\ell_X + \lambda$	tm	$t(1+\frac{m}{2})$	$(sm+1)^t \geqslant 2^\lambda$
	-	-	✓	✓	✓$_R$	-	$t(\ell_G + 3\lambda)$	$\ell_X + \lambda$	tm	$t(1+\frac{m}{2})$	$(m+1)^t \geqslant 2^\lambda$
	✓	-	✓	✓	✓$_R$	-	$t(\ell_G + 3\lambda)$	$s\ell_X + \lambda$	tm	$t(1+\frac{m}{2})$	$(sm+1)^t \geqslant 2^\lambda$
	-	✓	✓	✓	✓$_L$	-	$N_{\mathsf{seed}}3\lambda + w(\lceil\log(m)\rceil\lambda + \ell_G + 2\lambda)$	$\ell_X + \lambda$	tm	$t + w\frac{m+1}{2}$	$\binom{t}{w}m^w \geqslant 2^\lambda$
	✓	✓	✓	✓	✓$_L$	-	$N_{\mathsf{seed}}3\lambda + w(\lceil\log(m)\rceil\lambda + \ell_G + 2\lambda)$	$s\ell_X + \lambda$	tm	$t + w\frac{m+1}{2}$	$\binom{t}{w}(sm)^w \geqslant 2^\lambda$

Table 1: A summary of all meaningful combinations of techniques, with resulting maximum signature size, public-key size, signature time and verification time. The asterisks in the last row indicates a possibly expensive Merkle tree computation must be accounted for. As the Skipping Edges technique enjoys two variants (edges skipped on the left or on the right), the subscripts L and R in the **Skip** column denote which variant is considered. For more details on the parameters in the table, we refer to the preliminaries and the sections listed in the first column.

2 Preliminaries

In this section, we recall definitions and some properties of the various objects that will be at the center of our work.

2.1 Notation

We typically employ uppercase letters (e.g. A) for sets, groups etc. and lowercase letters (e.g. a) for their elements. We use serif (e.g. A) to represent formal objects such as parties, or algorithms, protocol components etc. The letter ℓ is used to denote the size (i.e. bit-length) of elements in a certain set; for example ℓ_X refers to the size of elements of a set X and ℓ_G to the size of elements of a group G. To ease notation, we simply write log instead of $\log_2$ to denote the logarithm in base 2. Finally, for notions that are relevant in cryptography, we adopt the usual choice of notation: λ represents the security parameter and $\xleftarrow{\$}$ denotes the act of sampling uniformly at random. Moreover, we assume the reader is familiar with standard cryptographic functions (e.g. hash, PRF, commitment functions).

2.2 Sigma Protocols and Digital Signatures

The main objects discussed in this work are *Sigma protocols*, i.e. 3-round interactive protocols between two parties, where the first party (the *prover*) has the goal of proving to the second party (the *verifier*) the knowledge of a *witness* w to a public *statement* x for a binary relation R.

Definition 1 (Σ-Protocol). *A Σ-protocol Π for a polynomially-computable binary relation R is defined by three sets* CmtSet, ChSet *and* RspSet *(respectively the* commitment *space,* challenge *space and* response *space) and two polynomial-time algorithms: a probabilistic, two-stage algorithm $\mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2)$, composed of a* commitment *routine and a* response *routine, and a deterministic* verification *algorithm V. When more than one Σ-protocol is discussed, we use a prefix (e.g. $\Pi.\mathsf{V}$) to clarify which protocol the various objects belong to. The flow of the protocol is as follows.*

Protocol 1 (Σ-Protocol)

1. *On input $(\mathsf{x}, \mathsf{w}) \in R$, the prover computes a commitment $\mathsf{cmt} \leftarrow \mathsf{P}_1(\mathsf{x}, \mathsf{w})$ and sends it to the verifier.*
2. *The verifier samples uniformly at random a challenge $\mathsf{ch} \in \mathsf{ChSet}$, and sends it to the prover.*
3. *Given ch, the prover computes a response $\mathsf{rsp} \leftarrow \mathsf{P}_2(\mathsf{x}, \mathsf{w}, \mathsf{cmt}, \mathsf{ch})$ and sends it to the verifier.*
4. *The verifier runs $\mathsf{V}(\mathsf{x}, \mathsf{cmt}, \mathsf{ch}, \mathsf{rsp})$ and outputs 1 (pass), or 0 (fail).*

A transcript $(\mathsf{cmt}, \mathsf{ch}, \mathsf{rsp}) \in \mathsf{CmtSet} \times \mathsf{ChSet} \times \mathsf{RspSet}$ *of the protocol is said to be valid (relative to the statement x) if $\mathsf{V}(\mathsf{x}, \mathsf{cmt}, \mathsf{ch}, \mathsf{rsp})$ outputs* 1.

We require the following properties of a Σ-protocol:

1. **Completeness.** All honestly generated transcripts must be valid. Thus:

$$\Pr\left[\mathsf{V}(\mathsf{x},\mathsf{cmt},\mathsf{ch},\mathsf{rsp})=1 \;\middle|\; \begin{array}{c}(\mathsf{x},\mathsf{w})\in R\\ \mathsf{cmt}\leftarrow\mathsf{P}_1(\mathsf{x},\mathsf{w}),\\ \mathsf{ch}\xleftarrow{\$}\mathsf{ChSet},\\ \mathsf{rsp}\leftarrow\mathsf{P}_2(\mathsf{x},\mathsf{w},\mathsf{cmt},\mathsf{ch})\end{array}\right]=1.$$

2. **Honest-Verifier Zero-Knowledge (HVZK).** Transcripts should be indistinguishable from random. More precisely, a Σ-protocol Π is *statistically HVZK* if there exists a probabilistic polynomial-time (PPT) simulator algorithm ZKSim such that, for any computationally unbounded adversary A, any $(\mathsf{x},\mathsf{w})\in R$ and $\mathsf{ch}\in\mathsf{ChSet}$, it holds

$$\Pr\left[\mathsf{A}(\mathsf{x},\mathsf{cmt},\mathsf{ch},\mathsf{rsp})=1 \;\middle|\; \begin{array}{l}\mathsf{cmt}\leftarrow\mathsf{P}_1(\mathsf{x},\mathsf{w});\\ \mathsf{rsp}\leftarrow\mathsf{P}_2(\mathsf{x},\mathsf{w},\mathsf{cmt},\mathsf{ch})\end{array}\right]-$$

$$\Pr\left[\mathsf{A}(\mathsf{x},\mathsf{cmt},\mathsf{ch},\mathsf{rsp})=1 \mid (\mathsf{cmt},\mathsf{rsp})\leftarrow\mathsf{ZKSim}(\mathsf{x},\mathsf{ch})\right]=\mathsf{negl}(\lambda)$$

   where λ is the security parameter. If the identity above above holds only for computationally-bounded adversaries, the protocol is instead said to be *computationally* HVZK.

3. **Special Soundness.** A malicious prover should have a bounded probability of passing verification. Formally, we say that Π is *special sound* if there exists a polynomial-time algorithm Ex, called *extractor*, such that, for any statement x and two valid transcripts (relative to x)

$$(\mathsf{cmt}\,,\,\mathsf{ch}\,,\,\mathsf{rsp})\,,\quad\text{and}\quad\left(\mathsf{cmt}\,,\,\mathsf{ch}'\,,\,\mathsf{rsp}'\right),$$

   with $\mathsf{ch}\neq\mathsf{ch}'$, it outputs a witness w to x. Special soundness implies that a malicious prover can make the verifier accept with probability at most

$$\varepsilon=\frac{1}{|\mathsf{ChSet}|}\tag{1}$$

   which is called *soundness error*.

To reduce the soundness error of a Σ-protocol Π, it is usual to make the challenge space ChSet larger using parallel repetitions. Formally speaking, this is a transformation of Π into a different Σ-protocol, as follows.

Transformation 1 (Parallel Repetitions)
This technique reduces the soundness error of a Σ-protocol via t independent repetitions.
Given: *a Σ-protocol Π.*
Transform: *a Σ-protocol Π' with $\Pi'.\mathsf{CmtSet} = (\Pi.\mathsf{CmtSet})^t$, $\Pi'.\mathsf{ChSet} = (\Pi.\mathsf{ChSet})^t$ and $\Pi'.\mathsf{RspSet} = (\Pi.\mathsf{RspSet})^t$. Given a statement x, a transcript for Π' is of the form*

$$\left(\mathsf{cmt}^{(i)}, \mathsf{ch}^{(i)}, \mathsf{rsp}^{(i)}\right)_{i=1,\cdots,t}$$

where each $(\mathsf{cmt}^{(i)}, \mathsf{ch}^{(i)}, \mathsf{rsp}^{(i)})$ is a transcript for Π. The algorithms are defined as follows. First, $\Pi'.\mathsf{P}_1$ and $\Pi'.\mathsf{P}_2$ simply execute $\Pi.\mathsf{P}_1$ and $\Pi.\mathsf{P}_2$ each t times, respectively; then, $\Pi'.\mathsf{V}$ checks the entire transcript and outputs 1 if and only if

$$\Pi.\mathsf{V}\left(\mathsf{cmt}^{(i)}, \mathsf{ch}^{(i)}, \mathsf{rsp}^{(i)}\right) = 1$$

for all $1 \leqslant i \leqslant t$.

It is easy to see that, if Π is complete, HVZK and special sound with soundness error ε, then so is Π'. In particular, the transform yields a soundness error $\varepsilon' = \varepsilon^t$.

The Fiat-Shamir Transformation. Any Σ-protocol Π can be made non-interactive using the Fiat-Shamir transformation [28]. In particular, the sampling of a uniformly-random challenge by the verifier (see step 2 in Protocol 1) is replaced by $\mathsf{ch} \leftarrow \mathsf{H}(\mathsf{cmt})$ for a collision-resistant hash function $\mathsf{H} : \{0,1\}^* \rightarrow \Pi.\mathsf{ChSet}$. This new protocol can in turn be transformed into a digital signature, which we will denote by $\mathsf{FS}(\Pi)$, by concatenating the message msg to be signed with cmt, in the evaluation of the hash function H. Before detailing the resulting signature scheme in Transformation 2, we recall the definition of digital signature.

Definition 2. *A digital signature scheme consists of three probabilistic polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ such that:*

- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda)$*: On input a security parameter λ, the key generation algorithm outputs a pair of verification and signing keys $(\mathsf{vk}, \mathsf{sk})$;*
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$*: On input a signing key sk and a message msg, the signing algorithm outputs a signature σ;*
- $1/0 \leftarrow \mathsf{Ver}(\mathsf{vk}, \mathsf{msg}, \sigma)$*: On input a verification key vk, a message msg and a signature σ, the verification algorithm outputs 1 (accept) or 0 (reject).*

We require a signature scheme to satisfy the following two properties and the security model which is commonly considered.

1. **Correctness.** For any security parameter $\lambda \in \mathbb{N}$ and any message msg the following holds:

$$\Pr\left[\mathsf{Ver}(\mathsf{vk}, \mathsf{msg}, \sigma) = 1 \,\middle|\, \begin{array}{c}(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda), \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})\end{array}\right] = 1.$$

2. **Unforgeability.** We define the *Existential Unforgeability under Chosen Message Attack (EUF-CMA)* via a game played by an adversary A and a challenger. Within the game, the challenger runs $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda)$ and provides the adversary A the verification key vk. It also prepares an empty set Q. The adversary A may adaptively submit messages msg to the challenger. Upon reception of msg, the challenger responds by returning $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$ to A and then updates the set $Q \leftarrow Q \cup \{\mathsf{msg}\}$. Finally, A outputs a forgery $(\mathsf{msg}^*, \sigma^*)$ and we say they win if $\mathsf{msg}^* \notin Q$ and $\mathsf{Ver}(\mathsf{vk}, \mathsf{msg}^*, \sigma^*) = 1$. A's advantage $\mathsf{Adv}_{\mathsf{A}}^{EUF-CMA}$ is defined as the probability they win the game. We say the signature scheme is $EUF-CMA$ secure if, for all probabilistic polynomial-time adversaries A, it holds that $\mathsf{Adv}_{\mathsf{A}}^{EUF-CMA}(\lambda) = \mathsf{negl}(\lambda)$, where again λ is the security parameter.

Transformation 2 (Fiat-Shamir)
This technique turns a Σ-protocol into a digital signature scheme.
Given: *a Σ-protocol Π and a hash function $\mathsf{H} : \{0,1\}^* \to \Pi.\mathsf{ChSet}$.*
Transform: *a digital signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ where*

- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\lambda)$*: On input a security parameter λ, the key generation algorithm samples a statement-witness pair $(\mathsf{x}, \mathsf{w}) \in R$, sets $(\mathsf{vk}, \mathsf{sk}) \leftarrow (\mathsf{x}, \mathsf{w})$ and returns $(\mathsf{vk}, \mathsf{sk})$;*
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$*: On input a signing key sk (with vk as corresponding verification key) and a message msg, the signing algorithm computes $\sigma_1 \leftarrow \mathsf{P}_1(\mathsf{vk}, \mathsf{sk})$, $\sigma_2 \leftarrow \mathsf{H}(\sigma_1, \mathsf{msg})$ and $\sigma_3 \leftarrow \Pi.\mathsf{P}_2(\mathsf{vk}, \mathsf{sk}, \sigma_1, \sigma_2)$, and outputs the signature $\sigma = (\sigma_1, \sigma_3)$;*
- $1/0 \leftarrow \mathsf{Ver}(\mathsf{vk}, \mathsf{msg}, \sigma)$*: On input a verification key vk, a message msg and a signature $\sigma = (\sigma_1, \sigma_3)$, the verification algorithm recomputes $\sigma_2 = \mathsf{H}(\sigma_1, \mathsf{msg})$ and then runs $\Pi.\mathsf{V}(\mathsf{vk}, \sigma_1, \sigma_2, \sigma_3)$.*

Remark 1. We note that the signature scheme in Transformation 2 can be immediately and straightforwardly optimised if Π is *commitment recoverable*. In fact, this property means that for any $(\mathsf{x}, \mathsf{w}) \in R$, any $\mathsf{ch} \in \Pi.\mathsf{ChSet}$ and any $\mathsf{rsp} \in \Pi.\mathsf{RspSet}$, there exists a unique commitment $\mathsf{cmt} \in \Pi.\mathsf{CmtSet}$ such that $(\mathsf{cmt}, \mathsf{ch}, \mathsf{rsp})$ is a valid transcript, and it can be publicly computed by means of a polynomial-time algorithm CmtRec taking $(\mathsf{x}, \mathsf{ch}, \mathsf{rsp})$ as input. The recovered commitment can be checked for equality against the challenge string in the verification phase, and so does not need to be included in the signature. In other words, we replace $\sigma = (\sigma_1, \sigma_3)$ with (σ_2, σ_3), and adapt the verification procedure accordingly, by recomputing σ_1 via CmtRec.

The resulting digital signature scheme is EUF-CMA secure, given that the challenge space is large enough [34].

Theorem 1. *Let* Π *be a* Σ*-protocol with challenge space* ChSet*, whose cardinality is exponential in the security parameter* λ*. Then the digital signature scheme* $\mathsf{FS}(\Pi)$ *is EUF-CMA secure.*

2.3 Cryptographic Group Actions

Let X be a set and $(G, \cdot)$ a group. A *group action* of G on X is a function of the form

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

which is *compatible* with the group operation, that is, $e \star x = x$ for all $x \in X$ for the neutral element $e \in G$, and $h \star (g \star x) = (h \cdot g) \star x$ for all $x \in X$ and all $g, h \in G$. A group action is 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 $g \in G \backslash \{e\}$ such that $x = g \star x$ for all $x \in X$;
- *free* if an element $g \in G$ is equal to e whenever there exists an $x \in X$ such that $x = g \star x$;
- *regular* if it is free and transitive.

The adjective *cryptographic* is used for group actions which enjoy additional properties that are relevant to cryptography [3, 15]. Namely, a cryptographic group action should be *one-way*, i.e., it should be easy to compute while, 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). This computational problem is known as the *vectorisation* problem, or as *Group Action Inverse Problem (GAIP)*.

Problem 1 (GAIP) *Given a group action* $\star : G \times X \to X$ *and uniformly-random elements* x *and* y *in* X*, find, if any, an element* $g \in G$ *such that* $y = g \star x$*.*

A related problem - equally required to be hard - asks to compute the action of the product of two group elements, given the the individual actions on a fixed element. This is known as the *parallelisation* problem and it essentially corresponds to the computational version of the Diffie-Hellman problem [23], formulated for generic group actions. A formal definition for the *computational Group Action Diffie-Hellman* problem is given below.

Problem 2 (cGADH) *Given a group action* $\star : G \times X \to X$*,* $x \in X$ *and* $g \star x$ *and* $h \star x$ *for uniformly-random elements* $g, h \in G$*, compute* $(g \cdot h) \star x$*.*

In fact, the analogy with the discrete-logarithm case is easily drawn by observing that this is simply the group action given by the exponentiation map on finite cyclic groups.

For cryptographic security, it may be required that group elements that fix set elements, i.e. stabilisers, are hard to find [10]. This gives the following problem.

Problem 3 (Stabiliser-Computation Problem) *Given a group action* $\star : G \times X \to X$*, a uniformly-random element* $x \in X$*, find* $g \in G \backslash \{e\}$ *such that* $g \star x = x$*, if such a* g *exists.*

Other cryptographic properties for group actions include those that make it *effective*, for example, the existence of efficient (probabilistic polynomial-time) algorithms for membership testing, sampling, and computation (of both the group operation and the group action). More on these properties can be found in [3]. Finally, more complex protocols often require a generalisation of GAIP to multiple elements x_i: the *Multiple Group Action Inverse Problem (mGAIP).*

Problem 4 (mGAIP) *Given a collection* $x_0, ..., x_{r-1}$ *in* X*, find, if any, an element* $g \in G$ *and two different indices* $j \neq j'$ *such that* $x_{j'} = g \star x_j$*.*

Clearly, solving GAIP allows to solve mGAIP. We show the equivalence between the two problems as Proposition 8 in Appendix D by showing the other direction, through a generalisation of Theorem 3 from [6].

2.4 Digital Signatures from Group Actions

Any cryptographic group action $\star : G \times X \to X$ can be used to construct a secure digital signature via the aforementioned Fiat-Shamir transform [28], with security based on GAIP. The main building block is a Σ-protocol Π for a relation $R \subset X \times X$, with $\mathsf{CmtSet} = X$, $\mathsf{ChSet} = \{0, 1\}$ and $\mathsf{RspSet} = G$. Given a statement x, i.e. a pair $(x, y) \in X \times X$, the goal is to prove knowledge of a witness $g \in G$ such that $y = g \star x$. That is done by committing to a random element $\tilde{x} = \tilde{g} \star x$, with $\tilde{g} \xleftarrow{\$} G$, and then disclosing the link between either $(x, \tilde{x})$ or $(y, \tilde{x})$.
The flow of the protocol is as follows.

Protocol 2 (Group Action Σ-Protocol)

1. *On input* $g, y = g \star x$*, the prover runs* P_1*, which samples* $\tilde{g} \xleftarrow{\$} G$ *and computes the commitment* $\mathsf{cmt} = \tilde{x} \leftarrow \tilde{g} \star x$*, then sends* cmt *to the verifier.*
2. *The verifier samples uniformly at random a challenge* $\mathsf{ch} \in \mathsf{ChSet}$*, and sends it to the prover.*
3. *Given* ch*, the prover runs* P_2 *to compute the response as follows. If* $\mathsf{ch} = 0$*, then* $\mathsf{rsp} \leftarrow \tilde{g}$*; else, if* $\mathsf{ch} = 1$ *then* $\mathsf{rsp} \leftarrow \tilde{g} g^{-1}$*. Then, the prover sends* rsp *to the verifier.*
4. *If* $\mathsf{ch} = 0$*, the verifies checks whether* $\mathsf{rsp} \star x = \mathsf{cmt}$*; else, if* $\mathsf{ch} = 1$ *checks whether* $\mathsf{rsp} \star y = \mathsf{cmt}$*. Finally, it outputs 1 (pass) or 0 (fail) accordingly.*

Remark 2. A crucial point is that, since $\tilde{g}$ is sampled uniformly at random, the response $\mathsf{rsp} = \tilde{g}$ for the case $\mathsf{ch} = 0$ can in practice be replaced by a (smaller) seed, of size λ, for a secure pseudorandom number generator PRF. Note, however, that it is possible to consider a variant where the uniform distribution for sampling $\tilde{g}$ is replaced by an alternative probability distribution on G. In transformations such as Parallel Repetitions, many seeds will end up being transmitted. It is possible to use a *Puncturable PRF* [14] to do this efficiently, and further compress the size of the resulting signature. We present an instance in Section 3.3, where we describe a transformation using the *seed tree*.

Remark 3. When length-λ seeds are employed, it is fundamental that the message is salted with a random length-2λ string salt. This is necessary to prevent attacks that can retrieve the witness after observation of $O(2^{\lambda/2})$ signatures [17]. It is fundamental that the salt is sampled uniformly at random, independently of the message-to-be-signed. Notice that this increases the signature size, as salt must be included in the signature.

Proposition 1. *Protocol 2 is complete, honest-verifier zero-knowledge and special sound.*

Proof. We go over each property one by one.
Completeness. A honest prover holding a private key g for the public key $y = g \star x$ always passes verification.
Honest-verifier zero-knowledge. Consider a simple simulator Sim that, on input $x \in X$ and a challenge ch, outputs the following.

- When $\mathsf{ch} = 0$, it samples $\tilde{h} \xleftarrow{\$} G$ and outputs $\mathsf{cmt} = \tilde{h} \star x$ and $\mathsf{rsp} = \tilde{h}$. The resulting transcript is exactly that of an honest execution of the protocol.
- When $\mathsf{ch} = 1$, it samples $\tilde{h} \xleftarrow{\$} G$ and outputs $\mathsf{cmt} = \tilde{h} \star y$ and $\mathsf{rsp} = \tilde{h}$. The resulting transcript leads to a distribution that is the same of the transcript distribution obtained by running honestly the protocol. In fact, because the multiplication by a group element determines a bijective map from G to G, both $\tilde{h}$ and $g^{-1}\tilde{g}$ have the same uniform distribution, thus $\tilde{h}$ and $g^{-1}\tilde{g}$ are indistinguishable also to an unbounded adversary.

Special soundness. Let $(\mathsf{cmt}, 0, \tilde{g})$ and $(\mathsf{cmt}, 1, \tilde{g}g^{-1})$ be two valid transcripts. Given $\tilde{g}$ and $\tilde{g}g^{-1}$, we get g^{-1}, and hence we can extract the witness g for y with respect to statement $((x, y), g)$ induced by GAIP. □

Thanks to the special soundness, and considering the size of the challenge set, the above protocol provides a soundness error of $1/2$. This can intuitively be amplified via Parallel Repetitions, setting $t = \lambda$. Once the soundness error is reduced to $2^{-\lambda}$, the protocol can be turned into a EUF-CMA-secure signature via Fiat-Shamir. Using Theorem 1 with a commitment recoverable protocol Π, the **average signature size** is given by

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{\frac{1}{2}\lambda}_{\mathsf{rsp\ for\ ch}=0} + \underbrace{\frac{1}{2}\ell_G}_{\mathsf{rsp\ for\ ch}=1} \Big) . \tag{2}$$

In practice, the **maximum signature size** is more relevant, as an implementation needs to allocate a certain amount of bits for a signature. In this instance, this is given by

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\ell_G \tag{3}$$

which corresponds to row 1 of Table 1. Whenever these values differ, we list both the average and maximum signature size. As this work is practice-focused, we list the maximum signature size in Table 1 to ease comparison between different techniques for real-world applications.

The first cryptosystem using this generic construction, even if only sketched, traces back to [37], while the first practical implementation is the SeaSign scheme [22], using the CSIDH group action [16]. It is worth noting that SeaSign also employ rejection-sampling techniques, because there is no efficient way to sample uniformly random group elements for the CSIDH group action. (i.e. realise $g \xleftarrow{\$} G$). The rejection discards all elements with ideal powers over a threshold, similarly to what is done in the CRYSTALS-Dilithium scheme [24]. Other recently-designed signatures based on this construction are [8, 11, 19].

3 Known Optimisation Techniques

In the previous section, we have described a generic construction for a digital signature scheme from a cryptographic group action. Several optimisation techniques can be applied on top of this construction, in all generality. In this section, we describe the most prominent techniques that have emerged in the literature.

The initial Σ-protocol (Protocol 2) has a public pair of nodes (x, y) and commits to a node $\tilde{x}$. The prover reveals how to compute $\tilde{x}$ either from x (i.e. via $\tilde{g}$) or from y (i.e. via $\tilde{g}g^{-1}$). To generalise this terminology into a unified exposition of techniques for Σ-protocols based on group actions, we will describe them using the language of *action graphs*.

Action Graphs. An action graph for a group action $\star : G \times X \to X$ is the graph where the nodes are elements of X, and an edge between x_1 and x_2 is present whenever $x_2 = g \star x_1$ for some $g \in G$. A specific protocol, then, can be represented via the corresponding *subgraphs*, consisting of the nodes and edges involved.

Note that the use of a graph to study signature schemes induced by group actions is not new: for example, in [12], this was used as a theoretical abstraction to model all the possible commitment strategies and bound the number of set elements present in the transcript for a fixed soundness error. In our work, instead, we use graphs in a constructive way, to define protocols with different use cases and confront them with respect to different benchmarks. For instance, Protocol 2 can be described via the following graph:

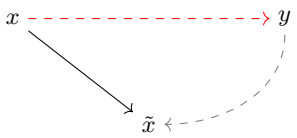


Fig. 1: Action graph for Protocol 2.

Let us clarify the meaning of the graph above. First of all, the public key is composed of an *origin* node x and a *destination* node y; the node $\tilde{x}$ is the randomly-generated *commitment*. The action of a group element on a node is seen as a map, and denoted via an arrow: for example, a dashed red arrow for public-key pairs, a solid arrow for commitments, and a gray dashed arrow for a map known only to those with the associated secret key. A challenge asks to reveal a specific path between two nodes: for example, $\mathsf{ch} = 0$ asks for the path between x and $\tilde{x}$, while $\mathsf{ch} = 1$ asks for the path between y and $\tilde{x}$. It follows, then, that responses would consist of paths, represent by the corresponding arrows: here, the solid line for the response to $\mathsf{ch} = 0$, and the dashed gray line for the response to $\mathsf{ch} = 1$.

An elaborate introduction to action graphs, including a general description of Σ-protocols, can be found in Appendix A.

Representations of Elements. The way that elements are represented is of significant importance to the bandwidth efficiency of signature schemes based on group actions.

- *Group elements* that are used to obtain commitments are always sampled uniformly at random from G, thus can be generated and represented by a seed of length λ.
- *Challenges* are usually simply represented as integers[1].
- *Responses* consist of various non-random group elements, corresponding to the matching path required. Efficient representations of such elements are an active area of research [18, 20, 36] to reduce signature sizes.

Presentation of Techniques. We present each technique in a *transform box* (as for Transformation 1). Each box contains a summary of the key information of the transformation. The rest of each subsection includes a graphical representation, the main features of the transformation, and a brief discussion on security.

[1] The integer 0 represents the challenge that asks to link each commitment to the origin x. This particular challenge is referred to as the *consistency check*, as it asks for the information to repeat the commitment generation.

3.1 Multiple Public Keys

De Feo and Galbraith [22] first presented the idea of using multiple public keys for group actions to decrease the signature size by reducing the soundness error. This transformation is meant to be applied *before* any iterations, i.e. to Protocol 2. By using multiple public keys $y_1, \ldots, y_s$ associated to different secret keys $g_1, \ldots, g_s$ such that $y_i = g_i \star x_i$, the challenge spaces increases from $\{0, 1\}$ to $\{0, \ldots, s\}$ as we now reveal a path from y_{ch} to the commitment $\tilde{x}$.

Transformation 3 (Multiple Keys)
This technique reduces the soundness error by increasing the number of public keys from only y to $y_1, \ldots, y_s$.
Given: *the Group Action Σ-Protocol Π.*
Transform: *the Σ-protocol Π' with $\Pi'.\mathsf{CmtSet} = \Pi.\mathsf{CmtSet}$, $\Pi'.\mathsf{RspSet} = \Pi.\mathsf{RspSet}$ and*

$$\Pi'.\mathsf{ChSet} = \{0, 1, ..., s\} \ .$$

The algorithms are defined as follows. First, we have $\Pi'.\mathsf{P}_1 = \Pi.\mathsf{P}_1$. Then, a challenge ch is sampled uniformly at random in $\Pi'.\mathsf{ChSet}$. If $\mathsf{ch} = 0$, then $\Pi'.\mathsf{P}_2 = \Pi.\mathsf{P}_2$; else, $\Pi'.\mathsf{P}_2$ computes rsp as $\tilde{g}g_{\mathsf{ch}}^{-1}$. Finally, $\Pi'.\mathsf{V}$ verifies validity as in $\Pi.\mathsf{V}$, but with respect to the statement x_{ch}.

Graphical Representation. The resulting action subgraph can be seen in Figure 2. By committing to a single set element $\tilde{x}$ we get $s + 1$ possible paths to reveal by composing the random path $x \to \tilde{x}$ with the secret paths $x \to y_j$.

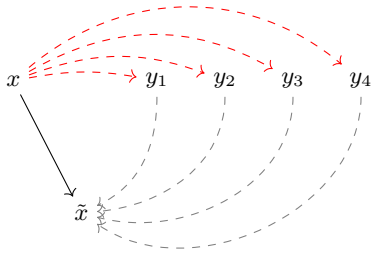


Fig. 2: Action subgraph with $s = 4$ public keys; the commitment $\tilde{x}$ can be linked to any of the public keys y_i.

Features. The transformation can be seen as a generalisation of Group Action Σ-Protocol, which corresponds to the special case $s = 1$. The transformation increases the size of the public key to $\lambda + s \cdot \ell_X$, while decreasing the **soundness error** to

$$\varepsilon' = \frac{1}{|\Pi'.\mathsf{ChSet}|} = \frac{1}{s+1} \ . \tag{4}$$

This means that, in principle, this optimisation could be used alone, without any iterations, if s was chosen large enough (e.g. $s+1 = 2^\lambda$). However, due to the rapid increase in public key size, this approach is de facto not applicable in practice. This optimisation is therefore viable when used together with multiple rounds. Namely, we can combine it with Parallel Repetitions, setting $t = \lceil \lambda / \log(s+1) \rceil$. Then, when transformed into a signature scheme via Fiat-Shamir, the resulting **average signature size** is given by

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{\frac{1}{s+1}\lambda}_{\mathsf{rsp\ for\ ch}=0} + \underbrace{\frac{s}{s+1}\ell_G}_{\mathsf{rsp\ for\ ch}\neq 0} \Big) \tag{5}$$

and the **maximum signature size** is therefore

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\ell_G \tag{6}$$

which corresponds to row 2 of Table 1. Hence, this optimisation yields a tradeoff between signature size and public key size. On the one hand, the number t of rounds required is now lower, thanks to the reduced soundness error, while on the other hand, the size of the public key grows linearly in s.

To decrease the combined size of public key and signature, the optimal value of s depends on the specific group action used, in particular on the balance between ℓ_X and ℓ_G. For schemes where set elements are relatively large, such as LESS or MEDS, the optimal value s is small (e.g. 4 or 5). Schemes with small set elements, such as SeaSign or CSI-FiSh, use a much larger number of public keys, compressed in a Merkle tree structure. For the response, only the set elements required for the challenges are transmitted (and verified with a Merkle proof). It should be noted that this strategy makes sense for the situations in which set elements and group elements are of comparable size; moreover, it still requires that all the public keys are computed during the key generation phase, so this should be considered as an additional limitation.

Security. The new protocol is complete, honest-verifier zero-knowledge and special sound. The proof is similar to the proof of Proposition 1, with the only exception that the soundness is now based on mGAIP, rather than GAIP.

3.2 Fixed-Weight Challenges

As seeds are a compact representation of randomly sampled objects, we try to represent objects by seeds whenever possible, while preserving the *preimage security* of the scheme. In practice, we accomplish this by sampling challenges from a skewed distribution, so that only a few challenges, say w, are non-zero, and the rest are zero. In other words, we sample the string of challenges as a vector of a pre-determined, fixed Hamming weight w (i.e. number of non-zero positions). Hence, this optimisation is commonly referred to as *Fixed-Weight Challenges* [7].

Transformation 4 (Fixed-Weight Challenges)
This technique increases the number of zero challenges by fixing the number w of non-zero challenges, thereby decreasing the communication cost.
Given: *a Σ-protocol Π.*
Transform: *a Σ-protocol Π' with $\Pi'.\mathsf{CmtSet} = (\Pi.\mathsf{CmtSet})^t$, $\Pi'.\mathsf{RspSet}' = (\Pi.\mathsf{RspSet})^t$ and*

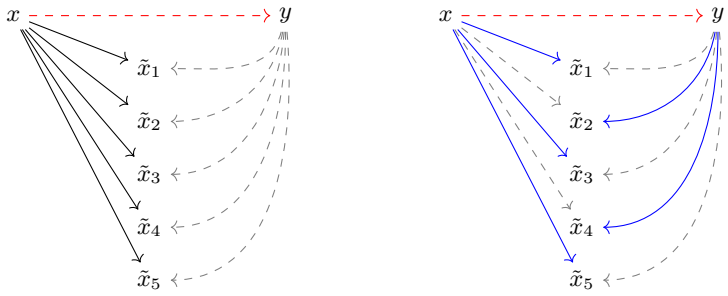
$$\Pi'.\mathsf{ChSet} = \{\ \mathsf{ch} = (\mathsf{ch}_1, \ldots, \mathsf{ch}_t) \in (\Pi.\mathsf{ChSet})^t \mid \mathsf{wt}(\mathsf{ch}) = w\ \}.$$

The algorithms are defined as follows. First, as in Parallel Repetitions, $\Pi'.\mathsf{P}_1$ and $\Pi'.\mathsf{P}_2$ simply execute $\Pi.\mathsf{P}_1$ and $\Pi.\mathsf{P}_2$ each t times, respectively; then, $\Pi'.\mathsf{V}$ checks the entire transcript and outputs 1 if and only if

$$\Pi.\mathsf{V}\left(\mathsf{cmt}^{(i)}, \mathsf{ch}^{(i)}, \mathsf{rsp}^{(i)}\right) = 1$$

for all $1 \leqslant i \leqslant t$.

Graphical Representation. An example of the resulting graph can be seen in Figure 3.



(a) Action graph with $t = 5$ rounds. (b) Response to $\mathsf{ch} = 01010$, with $w = 2$.

Fig. 3: Action graph with $\mathsf{ch}_i \in \{0, 1\}$ for $t = 5$ rounds and weight $w = 2$. Responses in blue, starting from x when $\mathsf{ch}_i = 0$, and from y when $\mathsf{ch}_i = 1$.

Features. The transform yields a **soundness error** of:

$$\varepsilon' = \frac{1}{|\Pi'.\mathsf{ChSet}|} = \binom{t}{w}^{-1}. \tag{7}$$

For each response, we send t group elements. Out of these, the $(t - w)$ group elements $\tilde{g}_i$ for $\mathsf{ch}_i = 0$ can be represented by a seed of size λ, whereas the

remaining w group elements $\tilde{g}_i g^{-1}$ for $\mathsf{ch}_i \neq 0$ are communicated as a full group element of size ℓ_G. After applying Fiat-Shamir, the **exact signature size** is given by

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + \underbrace{(t-w)\cdot\lambda}_{\mathsf{rsp\ for\ ch}_i=0} + \underbrace{w\cdot\ell_G}_{\mathsf{rsp\ for\ ch}_i\neq 0} \tag{8}$$

which corresponds to row 3 of Table 1. Thus, Fixed-Weight Challenges meaningfully decreases signature size whenever there is a noticeable asymmetry between λ, for $\mathsf{ch}_i = 0$, and ℓ_G for $\mathsf{ch}_i \neq 0$. To get λ-bit security, the values of t and w have to be set so that $\epsilon' \leqslant 2^{-\lambda}$. In particular, t is larger than the obtained via Parallel Repetitions.

Security. When Π is complete, honest-verifier zero-knowledge, and special sound, then so is Π'. The only difference with Parallel Repetitions is the form of the challenge string, which only requires adjusting the values of t and w.

3.3 Seed Trees

In this section, we describe the *seed tree* construction, which is a specific realisation of a *puncturable pseudorandom function (PRF)* [14]. In a nutshell, a puncturable PRF works by compressing the number of bits necessary to communicate many *seeded elements*, i.e. some pseudorandom data. In our use case, a seeded element is simply represented by a length-λ seed. For our purposes, we can define a puncturable PRF as a function PPRF taking as input an index $i \in \{1, \cdots, t\}$, where t is a positive integer, and a master seed $\mathsf{seed}_{\mathsf{root}} \in \{0,1\}^\lambda$, with the following properties:

1) for each $i \in \{1, \cdots, t\}$, the output of $\mathsf{PPRF}(i, \mathsf{seed}_{\mathsf{root}})$ is a length-λ binary string;
2) there exists an efficient algorithm $\mathsf{Recover}$ such that, for any index $i \in \{1, \cdots, t\}$, there exists some (efficiently-computable) binary string seeds_i^*, of size less than or equal to $(t-1)\lambda$, such that

$$\mathsf{Recover}(j, \mathsf{seeds}_i^*) = \mathsf{PPRF}(j, \mathsf{seed}_{\mathsf{root}}), \quad \forall j \neq i;$$

3) given access to i and seeds_i^*, the value of $\mathsf{PPRF}(i, \mathsf{seed}_{\mathsf{root}})$ must be indistinguishable from a uniformly-random binary length-λ string.

The definition can be extended by requiring that analogous properties hold when puncturing on a set of indices $\{i_1, \cdots, i_w\}$ of size $w > 1$, instead of a single one.

Since the seed tree is the best-known example of puncturable PRF (and the only one used so far when building signatures from Sigma protocols), we restrict our attention to it from now on. Assuming new (and, perhaps, better performing) examples of puncturable PRFs are found, these functions can be used to replace the seed trees we consider in our treatment.

On input a positive integer t and a root seed $\mathsf{seed}_{\mathsf{root}}$, a seed tree generates the required t seeds in a recursive way, constructing a binary tree. The tree is built by setting each node as a λ-bit string and using it to generate two child nodes using a secure function $\mathsf{PRF} : \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$, with the first λ bits of the output string being the left child, and the last λ being the right child. With this procedure, starting from a root seed $\mathsf{seed}_{\mathsf{root}}$, it is possible to obtain a complete binary tree with t leaves in the base layer whenever t is a power of 2. Each of these efficiently-generated leaves can then be used as a seed to obtain the various pseudorandom objects which are required to be computed in a protocol.
More formally, a seed tree is composed of three functions [7]:

- $\{\mathsf{leaf}_i\}_{i=1}^{t} \leftarrow \mathsf{SeedTree}\,(\mathsf{seed}_{\mathsf{root}}, t)$: On input a root seed $\mathsf{seed}_{\mathsf{root}} \in \{0,1\}^{\lambda}$ and a positive integer, constructs a complete binary tree with at least t leaves by recursively expanding each seed using PRF, as described above.
- $\mathsf{seeds}_{\mathsf{internal}} \leftarrow \mathsf{ReleaseSeeds}\,(\mathsf{seed}_{\mathsf{root}}, \mathsf{ch})$: On input a root seed $\mathsf{seed}_{\mathsf{root}} \in \{0,1\}^{\lambda}$, and a challenge $\mathsf{ch} \in \{0,1\}^{t}$, outputs a list of seeds $\mathsf{seeds}_{\mathsf{internal}}$ associated to nodes. These cover the set of leaves with index i satisfying $\mathsf{ch}_i = 0$. This means that the leaves contained in the subtrees rooted at each of the seeds which form $\mathsf{seeds}_{\mathsf{internal}}$ coincide with the leaves with i such that $\mathsf{ch}_i = 0$.
- $\{\mathsf{leaf}_i\}_{i \text{ s.t. } \mathsf{ch}_i=0} \leftarrow \mathsf{RecoverLeaves}\,(\mathsf{seeds}_{\mathsf{internal}}, \mathsf{ch})$: On input a list of seeds $\mathsf{seeds}_{\mathsf{internal}}$ and a challenge ch, it computes and outputs all the leaves covered by the nodes in the list, that is, those leaf_i with index i such that $\mathsf{ch}_i = 0$.

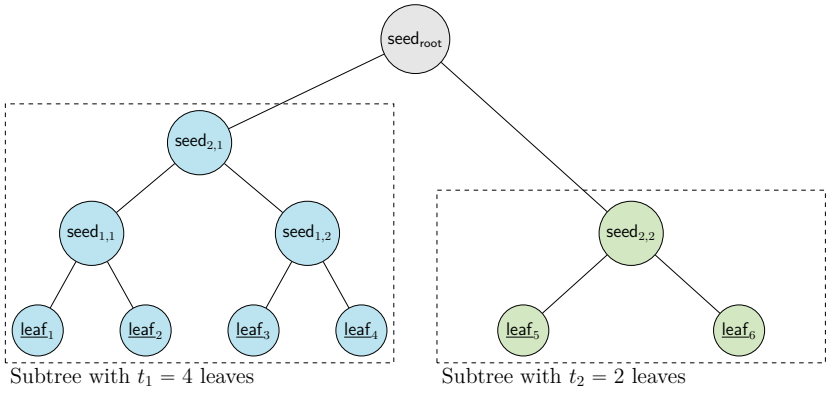
Note that, to ensure collision resistance of the expander function PRF, we need it to take as input $\mathsf{node}\|\mathsf{salt}\|\mathsf{idx}$, where salt is a length 2λ uniformly random sequence of bits (independent from the message) and idx is a unique predetermined index for the call, as explained in [17].

Transformation 5 (Seed Tree)
This technique reduces the communication cost by using a tree structure to generate seeds for pseudorandom objects.

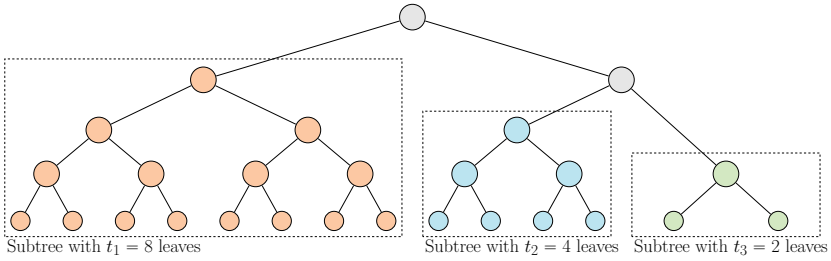
Given: *a Σ-protocol Π.*
Transform: *a Σ-protocol Π' with $\Pi'.\mathsf{CmtSet} = \Pi.\mathsf{CmtSet}$, $\Pi'.\mathsf{ChSet} = \Pi.\mathsf{ChSet}$ and $\Pi'.\mathsf{RspSet} = \Pi.\mathsf{RspSet}$.*

The algorithms are defined as follows. First, $\Pi'.\mathsf{P}_1$ generates a random $\mathsf{seed}_{\mathsf{root}}$ and invokes $\mathsf{SeedTree}\,(\mathsf{seed}_{\mathsf{root}}, t)$ to produce t leaves. These leaves are then used by $\Pi'.\mathsf{P}_1$ as pseudorandom strings to generate the commitment. Next, a challenge ch is sampled uniformly at random in $\Pi'.\mathsf{ChSet}$. Then, $\Pi'.\mathsf{P}_2$ computes $\mathsf{seeds}_{internal}$ using $\mathsf{ReleaseSeeds}\,(\mathsf{seed}_{\mathsf{root}}, \mathsf{ch})$ and transmits this list as part of the response. Finally, $\Pi'.\mathsf{V}$ recovers the relevant leaves via $\mathsf{RecoverLeaves}\,(\mathsf{seed}_{\mathsf{internal}}, \mathsf{ch})$ and verifies validity as in $\Pi.\mathsf{V}$.

Graphical Representation. When t is not an integer power of 2, there are several ways to build a seed tree. In this paper, we consider the construction reported in Figure 4. Namely, we write $t = t_1 + \cdots + t_u$, with each t_i being a power of 2 and such that $t_1 > t_2 > \cdots > t_u$. To each t_i, we associate a tree with t_i leaves. The individual trees are then merged, starting from the ones with t_{u-1} and t_u leaves.



(a) $t = 6$

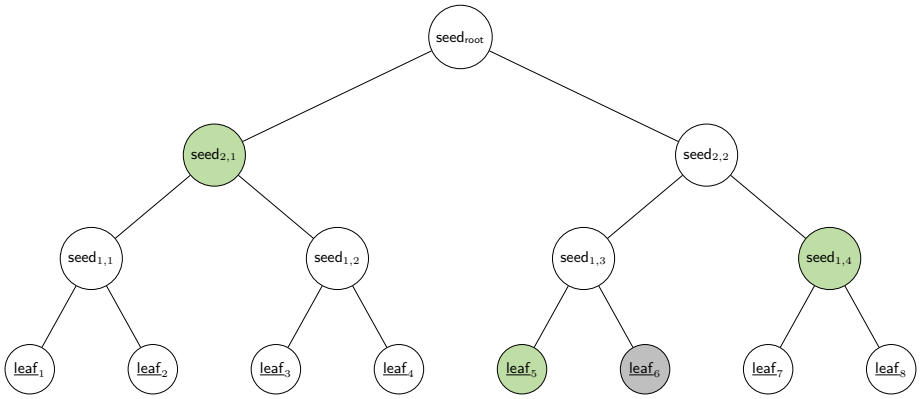


(b) $t = 14$

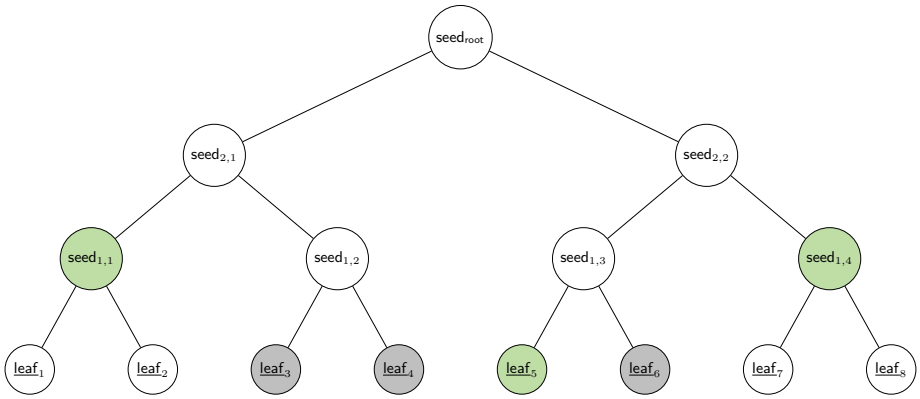
Fig. 4: Example of seed tree constructions, with t not being a power of 2.

Features. The effectiveness of using Seed Tree depends on several factors. First of all, note that the **soundness error** is unchanged with this transformation. The signature size is instead affected considerably, according to which scenario presents itself, as we explain next. For convenience, in Figure 5 we present an explicit visual description of the different scenarios for a seed tree.

First, we present our main result. For ease of the reader, we have included all the relevant intermediate results, and their corresponding proofs, in Appendix B.



(a) Example of a seed tree for $t = 8$ and $w = 1$



(b) Example of a seed tree for $t = 8$ and $w = 3$

Fig. 5: In this figure we can see 2 different possible scenarios for a seed tree. In every subfigure, the green nodes are used to recover all the leaves except for the grey ones.

Proposition 2. *Let $t = t_1 + t_2 + \cdots t_u$, where $t_1 > t_2 > \cdots > t_u$ and each t_i is an integer power of 2. Then, for any $\mathsf{ch} \in \{0,1\}^t$ with Hamming weight $w \leqslant t$, the output of* ReleaseSeeds *is not greater than*

$$N_{\mathsf{seeds}} = w \log(t/w) + u - 1 \ . \tag{9}$$

Proposition 2 gives an upper-bound to the size of ReleaseSeeds (i.e. the number of nodes required to recover the required leaves) for any fixed w and t. As a consequence of Proposition 4, for the special case in which we reveal all seeds but one, e.g. Figure 5a, the size can be bounded by

$$N_{\mathsf{seeds}} = \lceil \log(t) \rceil. \tag{10}$$

Remark 4. It is immediate to see that the same estimates from this section can be applied to round Merkle proofs for the instances in which we commit to t values and want to verify only w of them.

Lastly, we should also consider the case in which we reveal the first $i = t - w$ adjacent leaves out of t, i.e.

$$\mathsf{ch} = (\underbrace{1, ..., 1}_{i \text{ entries}}, 0, ..., 0) \ .$$

To deal with this case, we could consider again the binary tree construction described before; however, it is possible to get a better compression by considering a *sequential* puncturable PRF, that from an initial seed seed_t generates

$$\mathsf{seed}_j \leftarrow \mathsf{PRF}(\mathsf{seed}_{j+1} \| \mathsf{salt} \| \mathsf{idx}), \text{ for } j = t-1, ..., 1 \tag{11}$$

where salt is again a length-2λ uniformly random sequence of bits (independent from the message) and idx is a unique predetermined index for the call. In this way, thanks to the security of PRF, we can recover all seed_j for $j \leqslant i$, without leaking any information on the previous seeds, by just revealing seed_i (λ bits). Clearly, the reverse construction can be used to reveal the last i leaves out of t.

Remark 5. Analogously to seed tree and Merkle proof, we can also sequentially commit to all the t values $c_1, ..., c_t$, but verify only the first i one. Given a commitment function $\mathsf{C} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ we start from $\mathsf{cmt}_t \leftarrow \mathsf{C}(c_t)$, then:

$$\mathsf{cmt}_j \leftarrow \mathsf{C}(c_j \| \mathsf{cmt}_{j+1}), \text{ for } j = t-1, ..., 1 \tag{12}$$

so that the final committed value is $\mathsf{cmt} = \mathsf{cmt}_1$. It is clear that this value can be recomputed via $c_1, ..., c_i$ and cmt_{i+1}.

Security. Beullens, Katsumata, and Pintore [7, Lemma 1] show that, when modelling the function PRF as a random oracle, the seeds obtained by the function $\mathsf{ReleaseSeeds}$ using $\mathsf{seed}_{\mathsf{root}}$ are indistinguishable by the one generated via a simulator without access to $\mathsf{seed}_{\mathsf{root}}$ for any computationally unbounded adversary making up to a polynomial number of calls to the random oracle.

4 Novel and Recent Techniques

In the previous section, we have described the most well-known optimisations present in literature. In this section, we proceed instead to flesh out some other ideas that emerged recently, or have only recently been applied to the context of group actions, as well as new techniques and improvements.

4.1 MPC-in-the-Head for Group Actions

In the protocols described before we only perform walks of length one in the action graph, starting from the origin x or one of the public keys y_i. In this section (and subsequent ones) we will explore what protocols may arise by instead committing to a longer walk. In particular, such protocols are deeply linked with the MPC-in-the-head technique [30, 31].

Transformation 6 (MPC-in-the-Head)
This technique decreases the communication cost by walking a longer path in the action graph.
Given: *the Group Action Σ-Protocol* Π.
Transform: *the* Σ*-protocol* Π' *with* $\Pi'.\mathsf{CmtSet} = (\Pi.\mathsf{CmtSet})^m$, $\Pi'.\mathsf{ChSet} = \{0, ..., m\}$ *and* $\Pi'.\mathsf{RspSet} = (\Pi.\mathsf{RspSet})^{m+1}$.

The algorithms are defined as follows. First, $\Pi'.\mathsf{P}_1$ *generates random group elements* $\tilde{g}_i$ *for* $i = 1, ..., m$ *and computes:*

$$x \xrightarrow{\tilde{g}_1} \tilde{x}_1 \xrightarrow{\tilde{g}_2} \dots \xrightarrow{\tilde{g}_m} \tilde{x}_m \quad \text{and} \quad \tilde{x}_m \xrightarrow{g_\Delta} y.$$

It then commits to $\tilde{x}_i$ *for* $i = 1, ..., m$. *Next, a challenge* ch *is sampled uniformly at random in* $\{0, ..., m\}$. *If* $\mathsf{ch} \neq m$, *the prover leaves all* $\tilde{g}_i$ *untouched by setting* $g'_i = \tilde{g}_i$ *except for* $g'_{\mathsf{ch}+1}$, *which is set to* $\tilde{g}_{\mathsf{ch}+1} \cdots \tilde{g}_1 \cdot g^{-1}$, *which maps* $y \mapsto \tilde{x}_{\mathsf{ch}+1}$. *Then, the prover responds with* $(g'_1, \dots, g'_m)$. *In case* $\mathsf{ch} = m$, *the prover alters no* $\tilde{g}_i$ *and simply sends* $(\tilde{g}_1, \dots, \tilde{g}_m)$. *Using* $(g'_1, \dots, g'_m)$, *the verifier computes the path*

$$x \xrightarrow{g'_1} \tilde{x}'_1 \xrightarrow{g'_2} \dots \xrightarrow{g'_{\mathsf{ch}}} \tilde{x}'_{\mathsf{ch}} \ ;$$

With $g'_{\mathsf{ch}+1}$, *it computes* $y \to \tilde{x}_{\mathsf{ch}+1}$ *and uses* g'_j *with* $j > \mathsf{ch} + 1$ *to compute the remaining path*

$$y \xrightarrow{g'_{\mathsf{ch}+1}} \tilde{x}'_{\mathsf{ch}+1} \xrightarrow{g'_{\mathsf{ch}+2}} \dots \xrightarrow{g'_m} \tilde{x}'_m \ .$$

Finally, the verifier checks that all $\tilde{x}'_i$ *are the committed values* $\tilde{x}_i$.

Graphical Representation. Figures 6 and 7 show an example of the action subgraph derived from this technique.

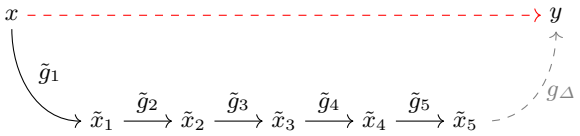


Fig. 6: The MPC-in-the-head action graph with $m = 5$ parties.

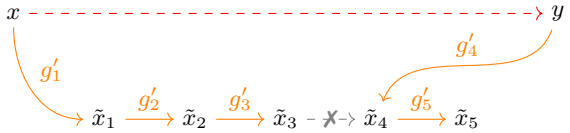


Fig. 7: For $\mathsf{ch} = 3$, we reveal the paths $x \to \tilde{x}_1 \to \tilde{x}_2 \to \tilde{x}_3$ and $y \to \tilde{x}_4 \to \tilde{x}_5$.

Features. This technique is associated with *MPC-in-the-Head* because the protocol can be seen as the simulation of $m+1$ users doing an MPC evaluation of the group action $g \star x$ by sequentially applying random group elements $\tilde{g}_i \in G$, until the last user applies $\tilde{g}_\Delta := g \cdot \tilde{g}_1^{-1} \cdots \tilde{g}_m^{-1}$ to get the random walk

$$x \xrightarrow{\tilde{g}_1} \tilde{x}_1 \xrightarrow{\tilde{g}_2} \ldots \xrightarrow{\tilde{g}_m} \tilde{x}_m \xrightarrow{\tilde{g}_\Delta} y \ .$$

The response simulates the revealing of all steps except step $\mathsf{ch}+1$, i.e. $\tilde{x}_{\mathsf{ch}} \xrightarrow{\tilde{g}_{\mathsf{ch}+1}} \tilde{x}_{\mathsf{ch}+1}$. Thus, the verifier recomputes the computations of all users except user $\mathsf{ch}+1$. For the transformed protocol the **soundness error** is

$$\varepsilon' = \frac{1}{|\Pi'.\mathsf{ChSet}|} = \frac{1}{m+1} \ . \tag{13}$$

For each response, we send m group elements. Out of these, $m-1$ group elements can be seeded (the unaffected g'_i with $i \neq \mathsf{ch}+1$). The remaining group element, the affected $\tilde{g}'_{\mathsf{ch}+1}$, is communicated as a full group element giving an upper bound[2] for the total communication cost as

$$\underbrace{(m-1)\lambda}_{\text{seeded } \tilde{g}_i} + \underbrace{\ell_G}_{\tilde{g}'_{\mathsf{ch}+1}} \ .$$

To achieve a λ-bit secure signature scheme, we repeat the protocol $t = \lceil \lambda / \log(m+1) \rceil$ times and apply Fiat-Shamir. The final **average signature size** is given by

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{\frac{1}{m+1}(m\lambda)}_{\mathsf{rsp\ for\ ch}=m} + \underbrace{\frac{m}{m+1}((m-1)\lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}\neq m} \Big) \ , \tag{14}$$

and thus, the **maximal signature size** becomes

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{(m-1)\lambda}_{\text{seeded } \tilde{g}_j} + \underbrace{\ell_G}_{g'_{\mathsf{ch}+1}} \Big) \ , \tag{15}$$

which corresponds to row 5 of Table 1. Both the generation and the verification of the signature requires

$$t \cdot m = \left\lceil \frac{\lambda}{\log(m+1)} \right\rceil \cdot m$$

group action evaluations. For large m, this quickly becomes costly: for most signature schemes based on group actions, the computational cost of verification is dominated by these group action evaluations. A detailed version of the protocol flow is given in Appendix C.

[2] The case $\mathsf{ch} = m$ uses only unaffected g'_i so can be communicated seeds for all g'_i.

Security. Joux [31, Thm. 2] shows that the unsalted and salted versions of MPC-in-the-Head are statistical honest-verifier zero-knowledge, with soundness error $1/m + \zeta$, where $1/\zeta$ is proportional to the runtime of the best adversary for GAIP.

4.2 Skipping Edges

With the MPC-in-the-Head technique, the soundness error is reduced to $1/m$ at the cost of computing, in each round, m group action evaluations. Both the prover and the verifier compute these evaluations: this is a common feature of all the techniques we saw, up to this point. By *skipping edges*, we show how to unbalance these costs. Namely, with a mild increase in the signature size, we can reduce the average number of group action evaluations for the verifier by a factor of two on average. This can yield a considerable speed-up in verification.

Transformation 7 (Skipping Edges)
This technique decreases the verification cost by combining several individual paths $\tilde{g}_j \in \mathsf{rsp}$ in the verification into a single path.
Given: *the MPC-in-the-Head protocol Π.*
Transform: *the Σ-protocol Π' with $\Pi'.\mathsf{CmtSet} = \{0,1\}^\lambda$, $\Pi'.\mathsf{ChSet} = \Pi.\mathsf{ChSet}$ and $\Pi'.\mathsf{RspSet} = \Pi.\mathsf{RspSet} \times \{0,1\}^*$.*

The protocol is sketched as follows. First, the prover generates m random group elements $\tilde{g}_1 \in G$ for $i = 1, ..., m$ and computes:

$$x \xrightarrow{\tilde{g}_1} \tilde{x}_1 \xrightarrow{\tilde{g}_2} \dots \xrightarrow{\tilde{g}_m} \tilde{x}_m \ .$$

It commits sequentially to $\tilde{x}_i$ for $i = 1, ..., m-1$ as in Remark 5 starting from $\tilde{x}_1$. Next, a challenge ch is sampled uniformly random in $\{0, ..., m\}$.

When $\mathsf{ch} \neq m$, the prover leaves all $\tilde{g}_i$ untouched by setting $g'_i = \tilde{g}_i$ except for $g'_{\mathsf{ch}+1}$, which is set to $\tilde{g}_{\mathsf{ch}+1} \cdots \tilde{g}_1 \cdot g^{-1}$, so that $\tilde{g}_{\mathsf{ch}+1}$ maps $y \mapsto \tilde{x}_{\mathsf{ch}+1}$. Then, the prover responds with $(g'_1, \dots, g'_{\mathsf{ch}+1}, \dots, g'_m)$.

When $\mathsf{ch} = m$, the prover alters no $\tilde{g}_i$ and simply sends $(\tilde{g}_1, \dots, \tilde{g}_m)$. When $\mathsf{ch} > 1$ the prover also adds the commitment $\mathsf{cmt}_{\mathsf{ch}-1}$ for the leaves $\tilde{x}_1, \cdots, \tilde{x}_{\mathsf{ch}-1}$. Using $(g'_1, \dots, g'_m)$, the verifier first computes $g'' = g'_{\mathsf{ch}} \ \cdots \ g'_1$ and computes the path

$$x \xrightarrow{g''} \tilde{x}'_{\mathsf{ch}} \ .$$

With $g'_{\mathsf{ch}+1}$, it computes $y \to \tilde{x}_{\mathsf{ch}+1}$ and uses g'_j with $j > \mathsf{ch}+1$ to compute the remaining path

$$y \xrightarrow{g'_{\mathsf{ch}+1}} \tilde{x}'_{\mathsf{ch}+1} \xrightarrow{g'_{\mathsf{ch}+2}} \dots \xrightarrow{g'_m} \tilde{x}'_m \ .$$

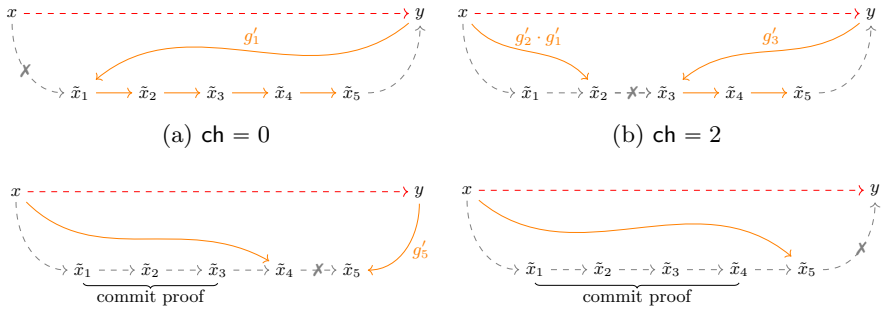
Finally, the verifier checks that all $\tilde{x}'_i$ are the committed values $\tilde{x}_i$ using Remark 5.

Remark 6. This description assumes we skip the *left* edges $\tilde{x}_i$ with $i < \mathsf{ch}$. One can also consider skipping the *right* edges $\tilde{x}_i$ with $i > \mathsf{ch} + 1$, mutatis mutandi, where the verifier computes instead:

$$x \xrightarrow{\tilde{g}'_1} \tilde{x}_1 \xrightarrow{\tilde{g}_2} \ldots \xrightarrow{\tilde{g}'_{\mathsf{ch}}} \tilde{x}'_{\mathsf{ch}} \quad \text{and} \quad y \xrightarrow{g'_{\mathsf{ch}+1}} \tilde{x}'_{\mathsf{ch}+1} \ .$$

Later we highlight the differences in these two approaches.

Graphical Representation. Figure 8 and Figure 9 show examples of action subgraphs for Skipping Edges. We visualise left skip in Figure 8, and right skip in Figure 9. The orange edges correspond to the group actions being evaluated, whereas the (unmarked) gray edges are the ones that we skip.These figures show how the number of group action evaluations changes depending on the challenge. For instance, in the *left skip* case, when $\mathsf{ch} = 0$ (visualised in Figure 8a), the verifier computes $m = 5$ group action evaluations to verify $y \to \tilde{x}_1 \to \ldots \to \tilde{x}_5$, hence no intermediate commitment value cmt_* is required for any $\tilde{x}_j$. For larger ch, the verifier computes fewer group action evaluations, as seen in Figures 8b to 8d, while $\tilde{x}_1, ..., \tilde{x}_{\mathsf{ch}-1}$ are verified with $\mathsf{cmt}_{\mathsf{ch}-1}$. A similar effect, but reversed, can be seen for the *right skip* case in Figure 9.



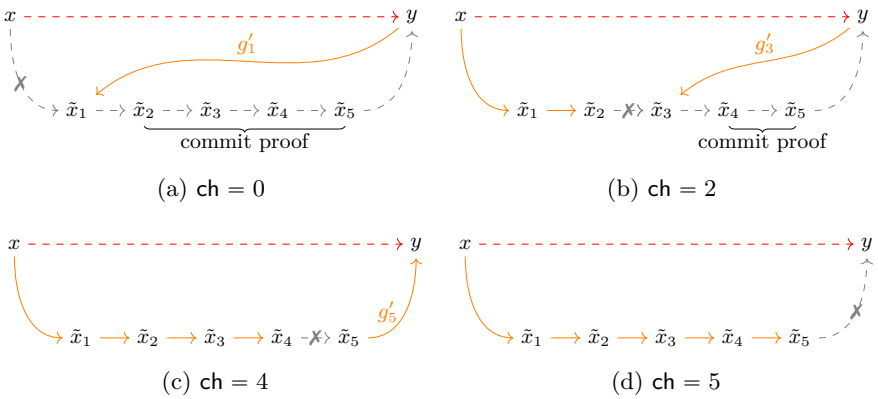
(a) $\mathsf{ch} = 0$ (b) $\mathsf{ch} = 2$

(c) $\mathsf{ch} = 4$ (d) $\mathsf{ch} = 5$

Fig. 8: Examples of responses for the MPC-in-the-head action graph with $m = 5$ parties, using Skipping Edges on *left side*.

Features. Compared to MPC-in-the-Head, the soundness error, public key size and signing time of Π' in comparison to Π does not change by any considerable factor, as the prover side only has negligible differences. Verification time, however, changes accordingly to ch: for the left skip variant, when $\mathsf{ch} = 0$, verification requires m group action evaluations, similar to MPC-in-the-Head. However, for $\mathsf{ch} > 0$, this reduces to $m + 1 - \mathsf{ch}$ group action evaluations. Thus, the **verification time**, on average, requires

$$\frac{1}{m+1}\Big(\underbrace{m}_{\mathsf{ch}=0} + \underbrace{\sum_{\mathsf{ch}=1}^{m} (m - \mathsf{ch} + 1)}_{\mathsf{ch}>0} \Big)$$



(a) $\mathsf{ch} = 0$ (b) $\mathsf{ch} = 2$

(c) $\mathsf{ch} = 4$ (d) $\mathsf{ch} = 5$

Fig. 9: Examples of responses for the MPC-in-the-head action graph with $m = 5$ parties, using Skipping Edges on *right side*.

group action evaluations. As this average is upper bounded by $1 + m/2$, we decrease from an average of m, for MPC-in-the-Head, by a factor of almost two. However, the verifier also computes $\mathsf{ch} - 1$ multiplications in the group to get $g'' = g'_{\mathsf{ch}} \cdots g'_1$. Phrased differently, the transformation turns $\mathsf{ch}-1$ group action evaluations into group multiplications. This optimisation therefore decreases the computation cost for verification whenever group multiplications are faster than group action evaluations, which is the case for most group actions.

Using Remark 5, the first $\mathsf{ch} - 1$ leaves can be verified with only 2λ bits in the communication cost. To achieve a λ-bit secure signature scheme, we repeat the protocol $t = \lceil \lambda / \log(m+1) \rceil$ times and apply Fiat-Shamir. The final **average signature size** for the *left skip* case is given by

$$\begin{aligned}\mathsf{SigSize}(\mathsf{FS}(\Pi')) &= \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + \\ &\quad + t\Big(\underbrace{\frac{1}{m+1}(m\lambda)}_{\mathsf{rsp\ for\ ch}=m} + \underbrace{\frac{m}{m+1}((m-1)\lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}\neq m} + \underbrace{2\lambda}_{\mathsf{cmt}_*} \Big) = \\ &\quad = \underbrace{\mathsf{SigSize}(\mathsf{FS}(\Pi))}_{\text{from (14)}} + t \cdot 2\lambda. \end{aligned} \tag{16}$$

The signature size is maximised when $\mathsf{ch} = m - 1$, which requires us to send all $\tilde{g}_i$ for $1 \leqslant i < m$ using a seed, and $\tilde{g}'_m$ as a full group element, and additionally the Merkle proof for all $\tilde{x}_i$ except $\tilde{x}_m$. Thus the **maximum signature size** is

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t \cdot \Big(\underbrace{(m-1)\lambda}_{\text{seeded } \tilde{g}_j} + \underbrace{\ell_G}_{g'_m} + \underbrace{2\lambda}_{\mathsf{cmt}_*} \Big), \tag{17}$$

which corresponds to row 6 of Table 1. When group elements are small, it is possible to replace $g'_1, \ldots, g'_{\mathsf{ch}}$ in the response by $g'' = g'_{\mathsf{ch}} \cdots g'_1$. However, in

many cases, the g'_i elements can be seeded compactly whereas communicating a full group element g'' is costly. Again a detailed version of the protocol flow is given in Appendix C.

For the *right skip* case the average verification time is reduced as before, and additionally we can avoid sending all $\tilde{g}_i$ for $i > \mathsf{ch} + 1$, since $g'_{\mathsf{ch}+1}$ is enough to compute the path $y \to \tilde{x}_{\mathsf{ch}+1}$. Thus, the communication cost of the responses for $\mathsf{ch} \neq m$ is, on average:

$$\frac{1}{m+1} \sum_{\mathsf{ch}=0}^{m-1} (\mathsf{ch}\lambda + \ell_G) \ . \tag{18}$$

With some algebraic manipulations, we can see that the final **average signature size** is

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + tm\Big(\frac{\ell_G}{m+1} + \frac{\lambda}{2}\Big) + t \cdot 2\lambda. \tag{19}$$

The worst case for the communications cost is $\mathsf{ch} = m - 1$, where we send all $\tilde{g}_i$ for $1 \leqslant i < m$ using a seed, and $\tilde{g}'_m$ as a full group element. Thus the **maximum signature size** is

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{\mathsf{MaxSigSize}(\mathsf{FS}(\Pi))}_{\text{from (15)}} = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t(\underbrace{(m-1)\lambda}_{\text{seeded } \tilde{g}_j} + \underbrace{\ell_G}_{g'_m}) \tag{20}$$

which corresponds to row 7 of Table 1.

Security. Assuming the hardness of GAIP, the protocol resulting from Skipping Edges is a secure zero-knowledge identification protocol, in the random oracle model, with soundness error $\frac{1}{m+1}$. This is a new result, not appearing before in the literature. We provide the theorem and proof in Appendix D.

4.3 Hypercube

When the group action is **commutative**, we can improve MPC-in-the-Head by reusing the same edges, i.e. group elements, after rearranging them. This saves both signing and verification time, if computing the group operation is much faster than computing a group action evaluation. What we present here is a generalisation of the *Hypercube MPC-in-the-Head* technique [2]. We illustrate it using an additive notation for the group operation, i.e. $g \star (h \star x) = (g + h) \star x$.

The core idea is simple. The scenario is similar to MPC-in-the-Head, that is, we have m randomly generated elements $\tilde{g}_i$. Thanks to commutativity, we can commit to combinations of $\sum \tilde{g}_i$ instead of only the m values $\tilde{g}_i \star x$, as long as we never reveal the full path $x \to y$ to the verifier in the response. We thus commit to the values

$$\tilde{x}_j := \Big(\sum_{i \in S_0(j)} \tilde{g}_i \Big) \star x, \quad 0 \leqslant j < m$$

where $S_0(j)$ denotes the integers from 0 to $m - 1$ with j-th bit equal to 0. The challenge is an n-bit string, where each bit j simulates a round of the Group

Action Σ-Protocol: when $\mathsf{ch}_j = 0$, the path $x \to \tilde{x}_j$ is revealed by $\sum_{i \in S_0(j)} \tilde{g}_i$, whereas when $\mathsf{ch}_j = 1$, we reveal the path $\tilde{x}_j \to y$ by $\tilde{g}_m + \sum_{i \in S_1(j)} \tilde{g}_i$. Here, $S_1(j)$ denotes the integers with j-th bit equal to 1, and $\tilde{g}_m$ plays a similar role to g_Δ in MPC-in-the-Head.

Transformation 8 (Hypercube)
This technique decreases signing and verification cost by rearranging paths in the graph.
Given: *the MPC-in-the-Head protocol* Π *with* $m = 2^n - 1$ *parties.*
Transform: *the* Σ*-protocol* Π' *with* $\Pi'.\mathsf{CmtSet} = \Pi.\mathsf{CmtSet}$, $\Pi'.\mathsf{RspSet} = \Pi.\mathsf{RspSet}$ *and*

$$\Pi' = \{0,1\}^n.$$

The algorithms are defined as follows. First, the prover generates m *random group elements* $\tilde{g}_0, \dots \tilde{g}_{m-1}$*, computes*

$$x \xrightarrow{\sum_{i \in S_0(j)} \tilde{g}_i} \tilde{x}_j \quad 0 \leqslant j < n,$$

and commits to all $\tilde{x}_j$*. Furthermore, the prover sets* $\tilde{g}_m = g - \sum_{i=0}^{m-1} \tilde{g}_j$*. Next, a challenge* ch *is sampled uniformly random in* $\{0,1\}^n$*. For each bit of the challenge, the prover sends* $\tilde{g}_i$ *for all* $i \in S_0(j)$ *if* $\mathsf{ch}_j = 0$*, and sends* $\tilde{g}_i$ *for all* $i \in S_1(j)$ *if* $\mathsf{ch}_j = 1$*. This allows a verifier to recompute* $\tilde{x}'_j$ *when* $\mathsf{ch}_j = 0$ *as*

$$x \xrightarrow{\sum_{i \in S_0(j)} \tilde{g}_i} \tilde{x}'_j \ ,$$

and when $\mathsf{ch}_j = 1$ *as*

$$y \xrightarrow{-\tilde{g}_m - \sum_{i \in S_1(d)} \tilde{g}_i} \tilde{x}'_j \ .$$

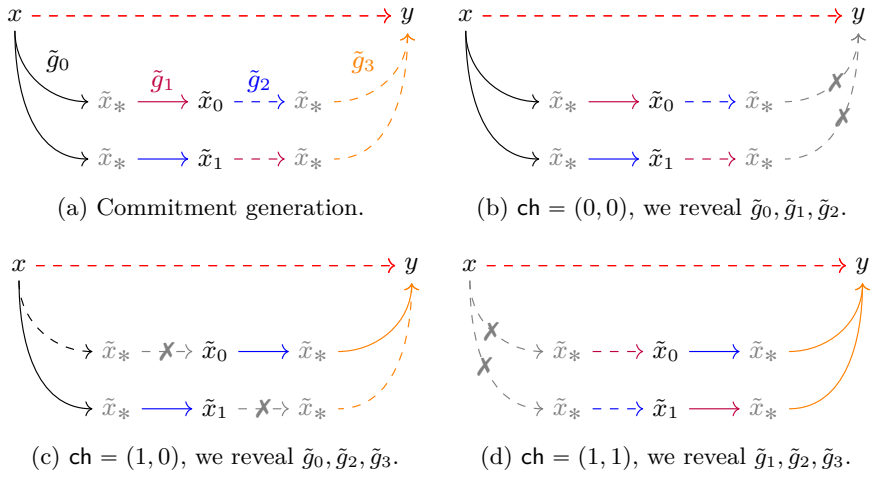
Finally, the verifier checks that all $\tilde{x}'_i$ *are the committed values.*

Graphical Representation. An example of how the Hypercube is applied for with $n = 2$ can be seen in Figure 10. Here we have $4 = 2^n$ parties that in Figure 10a, by rearranging the order of the group elements (distinguished by the colors of the arrows), commit to $\tilde{x}_0 = (\tilde{g}_0 + \tilde{g}_1) \star x$ and $\tilde{x}_1 = (\tilde{g}_0 + \tilde{g}_2) \star x$. The intermediate nodes $\tilde{x}_*$ in gray are not actually computed. In Figures 10b to 10d we see how, by sending all but one group elements, we can recompute the required paths using a share of them (dashed coloured lines are known but not used).

Features. The **soundness error** of the transformed protocol is the same as for MPC-in-the-Head,

$$\varepsilon = \frac{1}{|\Pi'.\mathsf{ChSet}|} = \frac{1}{2^n} = \frac{1}{m+1}.$$

Thus, we still need $t = \lceil \lambda / n \rceil$ rounds to reach λ-bit security.



(a) Commitment generation.

(b) $\mathsf{ch} = (0,0)$, we reveal $\tilde{g}_0, \tilde{g}_1, \tilde{g}_2$.

(c) $\mathsf{ch} = (1,0)$, we reveal $\tilde{g}_0, \tilde{g}_2, \tilde{g}_3$.

(d) $\mathsf{ch} = (1,1)$, we reveal $\tilde{g}_1, \tilde{g}_2, \tilde{g}_3$.

Fig. 10: Settings for the Hypercube MPCitH with $4 = 2^n = 2^2$ parties.

The response is composed of all group elements $\tilde{g}_j$ except for a single group element $\tilde{g}_k$ (the "gap"), where k is such that the j-th bit of k differs from each ch_j. In other words, k is the integer represented by flipping each bit of ch. Thus, the response contains $m-1$ randomly generated group elements $\tilde{g}_j$ and additionally the full group element $\tilde{g}_m$ required to compute paths $y \to \tilde{x}_j$. When $\mathsf{ch} = 1...1$, i.e. only 1s, the response contains just the random elements $\tilde{g}_j$, and $\tilde{g}_m$ is the "gap" that is left out. Thus, we get an **average signature size**, equal to m-party MPC-in-the-Head, of

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{\frac{1}{m+1}(m\lambda)}_{\mathsf{rsp\ for\ ch}=1...1} + \underbrace{\frac{m}{m+1}((m-1)\lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}\neq 1...1} \Big) \tag{21}$$

and similarly a **maximal signature size** of

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{(m-1)\lambda}_{\tilde{g}_j, \text{with } j\neq k} + \underbrace{\ell_G}_{\tilde{g}_m} \Big) \tag{22}$$

which corresponds to row 8 of Table 1.

The major difference between MPC-in-the-Head and Hypercube is in the signing and verification cost. Whereas MPC-in-the-Head needs m group action evaluations for commitment and verification, this reduces to $n = \log(m+1)$ using Hypercube, as we only need one group action evaluation per bit of the challenge. Hence, the signing and verification costs drop down to

$$t \cdot \log(m+1)$$

group action evaluations. However, both signer and verifier are required to compute $nm/2$ additions in the group, hence Hypercube mostly makes sense where

group operations are very fast compared to group action evaluations, as is the case in CSI-FiSH. Furthermore, as we need to generate $m = 2^n - 1$ random group elements, extreme parameters such as $n = \lambda$ are not feasible. A detailed version of the protocol flow is given in Appendix C.

Security. Given the similarity with the repeated version of the MPC-in-the-Head construction it is clear that soundness can be proved in the same way. However, part of the commitment and response are joined, hence we still need to verify the zero-knowledge property independently. This is done in Proposition 9 in Appendix D.

5 Combining Optimisation Techniques

Having presented both traditional and more recent transformations in Sections 3 and 4, this section describes how these individual optimisations combine together. We show the most common combinations seen in the current literature, as well as other efficient ones. The results in Table 1 are a concise summary of this section, and each subsection describes several rows of this table.

The general method is based on the three main transformations, combined with minor transformations. For each of these cases, we assume that the starting point is Protocol 2, plus the indicated transformation, organised as follows:

- Section 5.1 analyses combinations of Fixed-Weight Challenges with other transformations, giving rows 9 and 10.
- Section 5.2 analyses combinations of MPC-in-the-Head with other transformations, giving rows 11 to 16.
- Section 5.3 analyses the performance improvement gained from using Skipping Edges, whenever possible, giving rows 17 to 20.

From such a starting point, the other analysed transformations are applied sequentially (and interchangeably) until the final protocol is reached.

5.1 Fixed-Weight Challenges Combinations

We analyse combinations of Fixed-Weight Challenges with Seed Tree and Multiple Keys.

Seed Trees. Fixed-Weight Challenges naturally combines with Seed Tree to generate the t random group elements $\tilde{g}_i$ used in the commitment. This allows us to represent the $t - w$ elements for $\mathsf{ch}_i = 0$ in the response using the (small) list of seeds $\mathsf{seeds}_{\text{internal}}$ to generate precisely those leaves $\tilde{g}_i$. This combination leads to a **soundness error** of

$$\varepsilon' = \binom{t}{w}^{-1}. \tag{23}$$

Given t and w such that $\varepsilon' \leqslant 2^{-\lambda}$ and applying Fiat-Shamir, we get a **signature size** of

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + \underbrace{N_{\mathsf{seeds}}\lambda}_{\mathsf{rsp\ for\ ch}_i=0} + \underbrace{w \cdot \ell_G}_{\mathsf{rsp\ for\ ch}_i\neq 0} \tag{24}$$

where N_{seeds} are the number of nodes we need to release in the seed tree, from Equation (9). This equation corresponds to row 9 of Table 1.

Multiple Keys and Seed Trees. Additionally, one can also combine Fixed-Weight Challenges with Multiple Keys using s public keys, and exploiting Seed Tree to generate the t random group elements $\tilde{g}_i$ used in the commitment. We again represent the $t - w$ elements for $\mathsf{ch}_i = 0$ in the response using the (small) list of seeds $\mathsf{seeds}_{\text{internal}}$. This combination leads to a soundness error of

$$\varepsilon' = \binom{t}{w}^{-1} s^{-w} \ . \tag{25}$$

Given t, s and w such that $\varepsilon' \leqslant 2^{-\lambda}$ and applying Fiat-Shamir, we get a **signature size** of

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + \underbrace{N_{\mathsf{seeds}}\lambda}_{\mathsf{rsp\ for\ ch}_i=0} + \underbrace{w \cdot \ell_G}_{\mathsf{rsp\ for\ ch}_i\neq 0} \tag{26}$$

where N_{seeds} are the number of nodes we need to release in the seed tree from Equation (9). This equation corresponds to row 10 of Table 1. The combination of these three transformations is the most common in the current literature. For example, LESS [4, 6], MEDS [18, 19], and ALTEQ [11, 38] all use this combination, as described in more detail in Section 7.

5.2 MPC-in-the-Head Combinations

We analyse combinations of MPC-in-the-Head with the transformations from Sections 3 and 4.

Seed Trees. In Section 3.3, Equation (10) shows that, a seed tree with m leaves requires sending $\lceil \log(m) \rceil$ seeds to reveal all-but-one leaves. This is precisely the situation encountered in the MPC-in-the-Head, with or without Skipping Edges. More precisely, when $\mathsf{ch} \neq m$, we can represent the $m - 1$ random group elements $\tilde{g}_i$ using the $\lceil \log(m) \rceil$ seeds of size λ, and when $\mathsf{ch} = m$ we simply send the root of the tree. Thus, at the cost of a negligible overhead computation, we get a protocol Π' with $t = \lceil \lambda / \log(m+1) \rceil$ rounds to reach λ-bit security. After applying Fiat-Shamir, we get an **average signature size** of

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{\frac{1}{m+1}\lambda}_{\mathsf{rsp\ for\ ch}=m} + \underbrace{\frac{m}{m+1}(\lceil \log(m) \rceil \lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}\neq m} \Big) \tag{27}$$

and therefore a **maximum signature size of**

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t(\underbrace{\lceil \log(m) \rceil \lambda}_{\text{seeded } \tilde{g}_j} + \underbrace{\ell_G}_{g'_{\mathsf{ch}+1}}) \tag{28}$$

which corresponds to row 11 of Table 1.

Seed Trees and Multiple Keys. If, next to Seed Tree, we use Multiple Keys with s public keys (again with or without Skipping Edges), the challenge space per round becomes $\{0, ..., m-1\} \times \{1, ..., s\} \cup \{m\}$, and so, the soundness error is reduced from $\frac{1}{m+1}$ to $\frac{1}{sm+1}$. This reduces the number of rounds to t such that $(sm+1)^t \geqslant 2^\lambda$ and after applying Fiat-Shamir, the resulting **average signature size** becomes

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{\frac{1}{sm+1}\lambda}_{\mathsf{rsp\ for\ ch}=m} + \underbrace{\frac{sm}{sm+1}(\lceil \log(m) \rceil \lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}\neq m} \Big) \tag{29}$$

and therefore, a **maximum signature size** of

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t(\underbrace{\lceil \log(m) \rceil \lambda}_{\text{seeded } \tilde{g}_j} + \underbrace{\ell_G}_{g'_{\mathsf{ch}+1}}) \tag{30}$$

which corresponds to row 12 of Table 1. We are again in a particular case of the general Action Subgraph Σ-Protocol so the security is granted by Proposition 3.

Fixed-Weight and Seed Trees. By a similar logic as before, the large difference in communication cost between $\mathsf{ch} = m$ and $\mathsf{ch} \neq m$ allows us to apply Fixed-Weight Challenges to set the number of times $\mathsf{ch} = m$ occurs. Let Π' be the resulting protocol, where t denotes the number of rounds, and w the number of rounds with $\mathsf{ch}_i = m$; then the corresponding challenge set becomes

$$\{\ \mathsf{ch} = (\mathsf{ch}_1, \ldots, \mathsf{ch}_t) \in \{0, ..., m\}^t \mid \mathsf{wt}'(\mathsf{ch}) = w\ \}$$

where wt' is essentially the same as the Hamming weight, but counting the number of entries that are different from m, rather than 0. It follows that, using (7), the soundness error reduces to $\binom{t}{w}^{-1} m^{-w}$. We use Seed Tree to generate the $\tilde{g}_j$ per round, and another meta-Seed Tree to generate the root seeds $\mathsf{seed}_{\mathsf{root},i}$ of each seed tree per round. Whenever $\mathsf{ch}_i = m$, we only need to release the root seed $\mathsf{seed}_{\mathsf{root},i}$, hence we can disclose the $t - w$ leaves $\mathsf{seed}_{\mathsf{root},i}$ of the meta-seed tree using N_{seeds} nodes (see (9)) to release all the required $\tilde{g}_j$ for rounds i with $\mathsf{ch}_i = m$. Hence, given t, w, m such that $\varepsilon' \leqslant 2^{-\lambda}$ we get a **signature size** of

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + \underbrace{N_{\mathsf{seeds}}\lambda}_{\mathsf{rsp\ for\ ch}_i=m} + \underbrace{w(\lceil \log(m) \rceil \lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}_i\neq m}\ . \tag{31}$$

which corresponds to row 13 of Table 1. In both generation and verification, the number of group action evaluations increases to tm. By setting $m = 1$, we recover the combination from Section 5.1.

Fixed-Weight, Seed Trees, and Multiple Keys. If, next to Fixed-Weight Challenges and Seed Tree, we use Multiple Keys with s public keys, the challenge space per round is

$$\{0, ..., m-1\} \times \{1, ..., s\} \cup \{m\}$$

where the full challenge space requires w challenges with $\mathsf{ch} = m$. Since the protocol is still special sound by counting all the admissible sequence of challenges we see that the soundness error is reduced to $\binom{t}{w}^{-1}(sm)^{-w}$. Hence, given t, w, m, s such that $\varepsilon' \leqslant 2^{-\lambda}$, after applying Fiat-Shamir, we get a **signature size** of

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + \underbrace{N_{\mathsf{seeds}}\lambda}_{\mathsf{rsp\ for\ ch}_i=m} + \underbrace{w(\lceil \log(m) \rceil \lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}_i \neq m} . \tag{32}$$

which corresponds to row 14 of Table 1. Note that the difference between (31) and (32) lies in the improved soundness error obtained via multiple keys.

Hypercube and Seed Trees. Using Hypercube with t such that $(m+1)^t = 2^{nt} \geqslant 2^\lambda$, we can further rewrite Equation (28) to get an **average signature size** of

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{\frac{1}{m+1}\lambda}_{\mathsf{rsp\ for\ ch}=1\ldots1} + \underbrace{\frac{m}{m+1}(n\lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}\neq1..1} \Big) \tag{33}$$

and thus a **maximum signature size** of

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t(\underbrace{n\lambda}_{\tilde{g}_j, \text{with } j\neq k} + \underbrace{\ell_G}_{\tilde{g}_m}) \tag{34}$$

which corresponds to row 15 of Table 1. Note that for $n > 1$ we have $n = \lceil \log(m) \rceil$.

Hypercube, Seed Trees and Multiple Keys. Applying Multiple Keys with s public keys to the previous combination, the challenge space per round becomes $\mathsf{ChSet} = \{0,1\}^n$ to

$$\mathsf{ChSet}' = \underbrace{\{\mathsf{ch} \in \mathsf{ChSet} \mid \mathsf{ch} \neq (1, ..., 1)\}}_{m=2^n-1 \text{ elements}} \times \{1, ..., s\} \cup \{(1, ..., 1)\}.$$

Hence, the soundness error is reduced from $\frac{1}{m+1}$ to $\frac{1}{sm+1}$. It follows that, given t such that $(sm+1)^t \geqslant 2^\lambda$ and applying Fiat-Shamir, we obtain a signature scheme that performs a total of tn group action evaluations and yields an **average signature size**

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{\frac{1}{sm+1}\lambda}_{\mathsf{rsp\ for\ ch}=(1,\ldots,1)} + \underbrace{\frac{sm}{sm+1}(n\lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}\neq(1,\ldots,1)} \Big) \tag{35}$$

and consequently a **maximal signature size** of

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Big(\underbrace{n\lambda}_{\tilde{g}_j, \text{with } j \neq \mathsf{ch}} + \underbrace{\ell_G}_{\tilde{g}_{\mathsf{ch}}} \Big) \tag{36}$$

which corresponds to row 16 of Table 1.

5.3 Skipping Edges Combinations

We analyse the performance improvement when combining Skipping Edges with the previous techniques. We describe the *left skip* combinations first, and shortly discuss the *right skip* variants.

Seed Trees and Multiple Keys (left skip). We already pointed out that we can use both Seed Tree and Multiple Keys together with Skipping Edges, in the same way as usual. Given s public keys and m users for the *left skip* case, we obtain, as before, a soundness of $\frac{1}{1+sm}$ and an **average signature size** of

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + t\Bigg(\underbrace{\frac{1}{sm+1}\lambda}_{\mathsf{rsp\ for\ ch}=m} + \underbrace{\frac{sm}{sm+1}(\lceil \log(m) \rceil \lambda + \ell_G)}_{\mathsf{rsp\ for\ ch}\neq m} + \underbrace{2\lambda}_{\mathsf{cmt}_*} \Bigg) =$$

$$= \underbrace{\mathsf{SigSize}(\mathsf{FS}(\Pi'))}_{\text{from (29)}} + t \cdot 2\lambda \tag{37}$$

that is the same term as (29), plus the additional term due to the Merkle proof. By combining (30), we get a **maximum signature size**, independent of the number of public keys s, of

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = \underbrace{\mathsf{MaxSigSize}(\mathsf{FS}(\Pi))}_{\text{from (30)}} + t \cdot \underbrace{2\lambda}_{\mathsf{cmt}_*} \tag{38}$$

which corresponds to rows 17 and 18 of Table 1. The difference between using Multiple Keys or not is represented again by the trade off in the number of rounds t, and the size of the public key: in fact, for s public keys, the number of rounds is such that $(sm+1)^t \geqslant 2^\lambda$, thus decreases, but the public key size becomes $s\ell_X + \lambda$, hence increases.

Fixed-Weight, Seed Trees and Multiple Keys (left skip). When additionally applying the Fixed-Weight Challenges optimisation to the above, we get again the same improvements in soundness and signature size as in (32), with the additional cost of the Merkle proof. Using left skip, the number of group action evaluations during signing remains tm; however, the average number of group action evaluations during verification is reduced, via some algebraic manipulation, to

$$\underbrace{(t-w)}_{\mathsf{ch}=m} + \underbrace{w\left(\frac{1}{m} \sum_{\mathsf{ch}=0}^{m-1} (m+1-\mathsf{ch}) \right)}_{\mathsf{ch}=0,\ldots,m-1} = t + w \cdot \frac{m+1}{2} \ . \tag{39}$$

A direct comparison between the number of node commitments during signing and verification gives

$$\eta = \frac{t + w\frac{m+1}{2}}{tm} \approx \frac{1}{m} + \frac{w}{2t} < 1 \ . \tag{40}$$

We show in Section 6.3 that this leads to a reduction in the computational complexity for verification when compared not only to the combinations involving MPC-in-the-Head (Section 5.2), as seen in (40), but also to the combinations involving Fixed-Weight Challenges (Section 5.1).

To further reduce the signature size, we adapt the strategy used for committing to the intermediate nodes. For each round we commit to the first $m-1$ nodes $\tilde{x}_1, ..., \tilde{x}_{m-1}$ using again Remark 5 and obtain cmt_{m-1}, while the last one $\tilde{x}_m$ is committed separately as $\mathsf{cmt}_m = \mathsf{C}(\tilde{x}_m)$. The values cmt_{m-1} obtained in the t rounds are again committed via *another* Merkle Tree with t leaves. The final commitment cmt is derived from the root of this tree and the other t round values cmt_m.

Recall that the weight w in this case is defined as the number of rounds with $\mathsf{ch}_j \neq m$, i.e. the rounds in which we recompute cmt_{m-1}. For the remaining $t-w$ rounds, we instead use a cover for the binary tree as described in Remark 4, so N_{seeds} nodes are enough to verify the m rounds (see (9)). In the end, we get a **signature size** of

$$\begin{aligned}\mathsf{SigSize}(\mathsf{FS}(\Pi')) = \underbrace{2\lambda}_{\mathsf{salt}} + \underbrace{\lambda}_{\sigma_2} + \underbrace{N_{\mathsf{seeds}}\lambda}_{\mathsf{rsp}\text{ for }\mathsf{ch}_i=m} + \underbrace{N_{\mathsf{seeds}}2\lambda}_{\text{Merkle proof for }\mathsf{ch}_i=m} + \\ + w(\underbrace{\lceil\log(m)\rceil\cdot\lambda + \ell_G}_{\mathsf{rsp}\text{ for }\mathsf{ch}_i\neq m} + \underbrace{2\lambda}_{\mathsf{cmt}_*\text{ for }\mathsf{ch}_i\neq m}) = \\ = \underbrace{\mathsf{SigSize}(\mathsf{FS}(\Pi))}_{\text{from (32)}} + (N_{\mathsf{seeds}} + w)\cdot 2\lambda \ .\end{aligned} \tag{41}$$

which corresponds to rows 19 and 20 of Table 1.

Seed Trees and Multiple Keys (right skip). For the *right skip* variant with m users we can clearly use Multiple Keys with s public keys and the sequential PPRF from Section 3.3 described in (11). This way we obtain, as before, a soundness of $\frac{1}{1+sm}$ and an **average signature size** of

$$\mathsf{SigSize}(\mathsf{FS}(\Pi')) = t\left\{\underbrace{\frac{sm}{sm+1}}_{\mathsf{ch}\neq m}\left[\underbrace{\lambda}_{\text{seq. PPRF}} + \underbrace{\ell_G}_{g'_{\mathsf{ch}+1}} + \underbrace{2\lambda}_{\mathsf{cmt}_*}\right] + \underbrace{\frac{1}{sm+1}}_{\mathsf{ch}=m}\lambda\right\} . \tag{42}$$

By considering only the most expensive challenges, we get a **maximum signature size** of

$$\mathsf{MaxSigSize}(\mathsf{FS}(\Pi')) = t\cdot\big(\ell_G + 3\lambda\big) \tag{43}$$

which corresponds to rows 21 and 22 of Table 1. The number of round t is again such that $(1+sm)^{-t} \leqslant 2^{-\lambda}$.

Fixed-Weight (right skip). When applying Fixed-Weight Challenges to the right skip variant we encounter a problematic trade-off between signature size and verification time.

- when $\mathsf{ch}_i = m$, we send only the root of the seed tree, but we need to perform m group action evaluations in verification;
- when $\mathsf{ch}_i = 0$, we send at least one full group element, and perform only 1 group action evaluations in verification;
- when $0 < \mathsf{ch}_i < m$, we send one full group element, plus 3λ bits of additional information, and perform $\mathsf{ch}_i + 1 \leqslant m$ group action evaluations in verification.

This trade off makes the use of Fixed-Weight Challenges counterproductive with this variant. In fact, the main advantage would consist of decreasing as much as possible the ratio w/t of rounds with lightweight representations, i.e. the ones with $\mathsf{ch}_i = m$, since all the others requires at least one non-seeded group element. However, this would actually imply an increase in the number of group action evaluations, since now at least $m(t - w)$ of them are required. It follows that one cannot get optimal signature size while simultaneously reducing verification time, as in the *left skip* case. Due to this stalemate, this combination is best avoided.

6 Comparison between Optimisation Techniques

In this section, we scrutinise the effectiveness of transformations from Sections 3 to 5 from different points of view, showing (quite remarkably) that some of them are incompatible, or superseded in performance by combinations of other transformations. These general results offer to cryptographic designers clear and substantiated indications on which (combinations of) transformations using in order to make their scheme more efficient.

As a first result, we show in Section 6.1 that, in practice, Fixed-Weight Challenges is almost always better than MPC-in-the-Head, when used in combination with Seed Tree. We then examine this comparison when considering the more general construction of puncturable PRFs in Section 6.2. Finally, we show that Skipping Edges, combined as in Section 5.3, *can* improve on the current state of the art for group action-based signatures by decreasing the verification cost in comparison to Fixed-Weight Challenges.

6.1 Fixed-Weight Challenges vs MPC-in-the-Head

To compare Fixed-Weight Challenges and MPC-in-the-Head we provide a three-step argument. We give some high-level intuition by comparing Π_1, obtained from a combination of MPC-in-the-Head and Fixed-Weight Challenges with m parties and t_1 rounds with m-weight w_1, with Π_2, which *only* uses Fixed-Weight Challenges with $t_2 = t_1 m$ rounds with 0-weight w_2.

1. We show that, when using only a single public key, the challenge space of Π_1 can be interpreted as a subset of the challenge space of Π_2, even though Π_1 and Π_2 have both the same response size and the same computational cost.
2. We formalise this intuition in Theorem 2 for any number of public keys, by direct computation of the response sizes, computational complexity, and challenge spaces of Π_1 and Π_2.
3. We remove Fixed-Weight Challenges, i.e. we consider a pure MPC-in-the-Head protocol Π_1, and show in Theorem 3 that there always exists a protocol Π_2 using *only* Fixed-Weight Challenges with similar computational cost and response size, but a larger challenge space than Π_1.

High-level Intuition. Let us consider the protocol Π_1 using MPC-in-the-Head with m parties, combined with the Fixed-Weight Challenges optimisation with t_1 rounds, and let w_1 be the protocol's weight[3]. For simplicity, we start with the case of $s = 1$ public keys. For each round i, we denote the randomly-generated group elements by $\tilde{g}_1^{(i)}, \ldots, \tilde{g}_m^{(i)}$. Then, the challenge vector $\mathsf{ch}^{(1)}$ consists of t_1 integers $\mathsf{ch}_i^{(1)} \in \{1, \ldots, m\}$ such that $\mathsf{ch}_i^{(1)} = m$ for exactly $t_1 - w_1$ indices, and $\mathsf{ch}_i^{(1)} \neq m$ for exactly w_1 indices. In this case, the response consists of $t_1 \cdot m$ group elements, with all but w_1 generated by a seed, precisely those $\tilde{g}_j^{(i)}$ where $\mathsf{ch}_i^{(1)} = j \neq m$.

Consider now the protocol Π_2, using only Fixed-Weight Challenges with $t_2 = t_1 m$ rounds and weight[4] $w_2 = w_1$. For each round i, we have a single randomly generated $\tilde{g}_i$ and the challenge vector $\mathsf{ch}^{(2)}$ is a string of $t_2 = t_1 m$ bits with $\mathsf{ch}_i^{(2)} = 0$ for $t_2 - w_2$ indices, and $\mathsf{ch}_i^{(2)} = 1$ for w_2 indices. Thus, we can interpret $\mathsf{ch}^{(2)}$ also as a subset of $\{1, \cdots, t_2\} = \{1, \cdots, t_1 m\}$, of size $w_2 = w_1$, indicating the non-zero rounds. In this case, the response consists of $t_2 = t_1 m$ group elements, with $t_2 - w_1 = t_1 m - w_1$ generated by a seed, and w_2 described by a full group element.

Using a slightly different representation, every challenge for protocol Π_1 can be interpreted as a challenge for protocol Π_2. Let $\mathsf{ch}^{(1)}$ be a challenge for Π_1 and write each round challenge $\mathsf{ch}_j^{(1)} \in \{1, ..., m\}$ as a binary vector of length m with a 1 in position $\mathsf{ch}_j^{(1)}$ and 0 in all other positions. Hence, the full challenge $\mathsf{ch}^{(1)}$ becomes a binary vector $\mathsf{ch}^{(2)}$ of length $t_1 m$, subdivided into t_1 chunks of size m and weight 1, yielding a total (Hamming) weight of w_1. Then, $\mathsf{ch}^{(2)}$ is a valid challenge for Π_2 with weight $w_2 = w_1$. See Figure 11 for a graphical representation of the situation.

[3] Recall that here the word "weight" refers to the number of challenges that are different from m.

[4] In this instance, this is exactly the Hamming weight, thus counting the non-zero challenges.

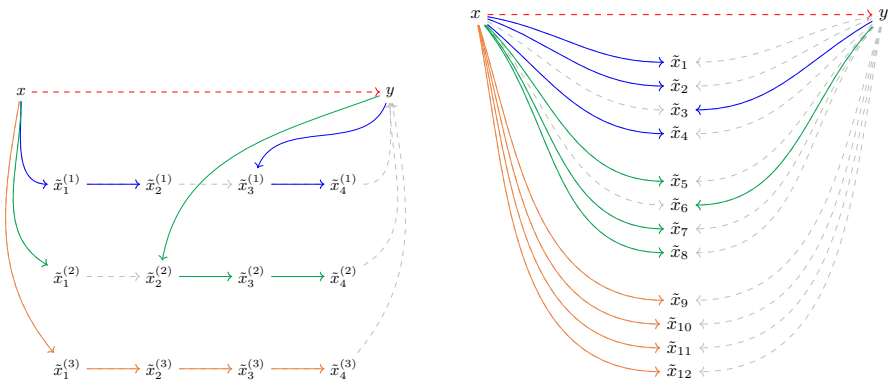


Fig. 11: Visualisation of challenges for protocols Π_1 and Π_2. Figure 11a shows an example for Π_1, which uses MPC-in-the-Head and Fixed-Weight Challenges, with $t_1 = 3$ rounds, $m = 4$ parties and weight $w_1 = 2$. Figure 11b shows the *equivalent* example for the protocol Π_2, which uses Fixed-Weight Challenges, with $t_2 = t_1 m = 12$ rounds subdivided into $t_1 = 3$ 'meta'-rounds of $m = 4$ rounds, with an overall weight $w_2 = 2$.

This reasoning shows that the challenge space $\Pi_1.\mathsf{ChSet}$ can be interpreted as a subset of the challenge space $\Pi_2.\mathsf{ChSet}$. However, the latter contains *all* possible vectors of length $t_1 m$ with weight w_1, whereas the former contains only those vectors having exactly t_1 chunks of length m and weight 1. Hence, Π_2 has a larger challenge space. Furthermore, both protocols have the same response size and are expected to have the same computational cost, as in both cases $t_1 m = t_2$ group actions are evaluated by both the prover and the verifier.

Formalising the High-level Intuition. We formalise this intuition in the following theorem, addressing also the more general case of $s > 1$.

Let Π_1 denote a protocol obtained using a combination of Multiple Keys, MPC-in-the-Head and Fixed-Weight Challenges, with s_1 public keys, m parties, and t_1 rounds, out of which w_1 rounds have a non-zero challenge, and let Π_2 denote the protocol obtained using Multiple Keys and Fixed-Weight Challenges, with $s_2 = s_1$ public keys, $t_2 = t_1 m$ rounds, out of which $w_2 = w_1$ rounds have a non-zero challenge.

Theorem 2. *For any tuple of values s_1, m, t_1 and w_1, with $m \geqslant 2$ and $t_1 \geqslant w_1$, the following holds.*

1. *The response sizes of Π_1 and Π_2 are equal.*
2. *The computational costs of Π_1 and Π_2 are equal.*
3. *The challenge space $\Pi_1.\mathsf{ChSet}$ is smaller than $\Pi_2.\mathsf{ChSet}$.*

Proof. We explicitly compute the three listed quantities for both Π_1 and Π_2.

Response size. For protocol Π_1, the response size $|\mathsf{rsp}^{(1)}|$ is

$$\begin{aligned}|\mathsf{rsp}^{(1)}| &= w_1\big((m-1)\lambda + \ell_G\big) + (t_1 - w_1)m\lambda \\ &= (t_1 m - w_1)\lambda + w_1\ell_G \\ &= |\mathsf{rsp}^{(2)}|\end{aligned}$$

where $|\mathsf{rsp}^{(2)}|$ denotes the response size for Π_2.

Computational cost. In both protocols, both the signer and verifier compute $t_1 m$ group action evaluations[5] to derive the $\tilde{x}_i$.

Challenge space. For the challenge spaces, we get

$$\begin{aligned}|\Pi_2.\mathsf{ChSet}| = \binom{t_2}{w_2} s_2^{w_2} = \binom{t_1 m}{w_1} s_1^{w_1} &= s_1^{w_1} \prod_{i=0}^{w_1-1} \frac{t_1 m - i}{w_1 - i} \\ &> s_1^{w_1-1} \prod_{i=0}^{w_1-1} \frac{m(t_1 - i)}{w_1 - i} = \binom{t_1}{w_1} m^{w_1-1} s_1^{w_1-1} \\ &= |\Pi_1.\mathsf{ChSet}|\end{aligned}$$

where the crucial observation is simply that $m(t_1 - i) \leqslant t_1 m - i$. □

The Impact on Signature Schemes. When considering signature schemes, the protocols that get transformed via Fiat-Shamir must have a challenge space of size at least 2^λ. As a consequence of Theorem 2, we have that $\mathsf{FS}(\Pi_1)$ always results in a signature scheme whose performance is not better than that of $\mathsf{FS}(\Pi_2)$. This result can be interpreted in different ways, depending on which performance aspect is considered. For instance, to have the same challenge space size when $w_2 = w_1$, Π_1 requires a larger complexity (i.e. $mt_1 > t_2$); equivalently, if the two protocols have the same computational complexity (i.e. $t_2 = mt_1$), then we can choose $w_2 < w_1$, which means that $\mathsf{FS}(\Pi_2)$ ultimately has shorter signatures (since the number of non ephemeral group elements is smaller).

To visualize these considerations, see Figure 12, where we report, as a function of $w = w_1 = w_2$, the minimum number of group action evaluations to achieve a challenge space of size at least 2^λ, for $\lambda = 128$. For both Π_1 and Π_2, we have computed the smallest values of t_1 and t_2 yielding a desired challenge space size; for Π_1 the number of computations is mt_1, while for Π_2 is t_2. The figure highlights the fact that Π_2 is always a preferable choice since it always leads lower complexities.

[5] We do not take parallelisation into account for the computational cost. As Π_1 computes its commitments per round in series, parallelisation improves the performance of Π_2 more than Π_1.

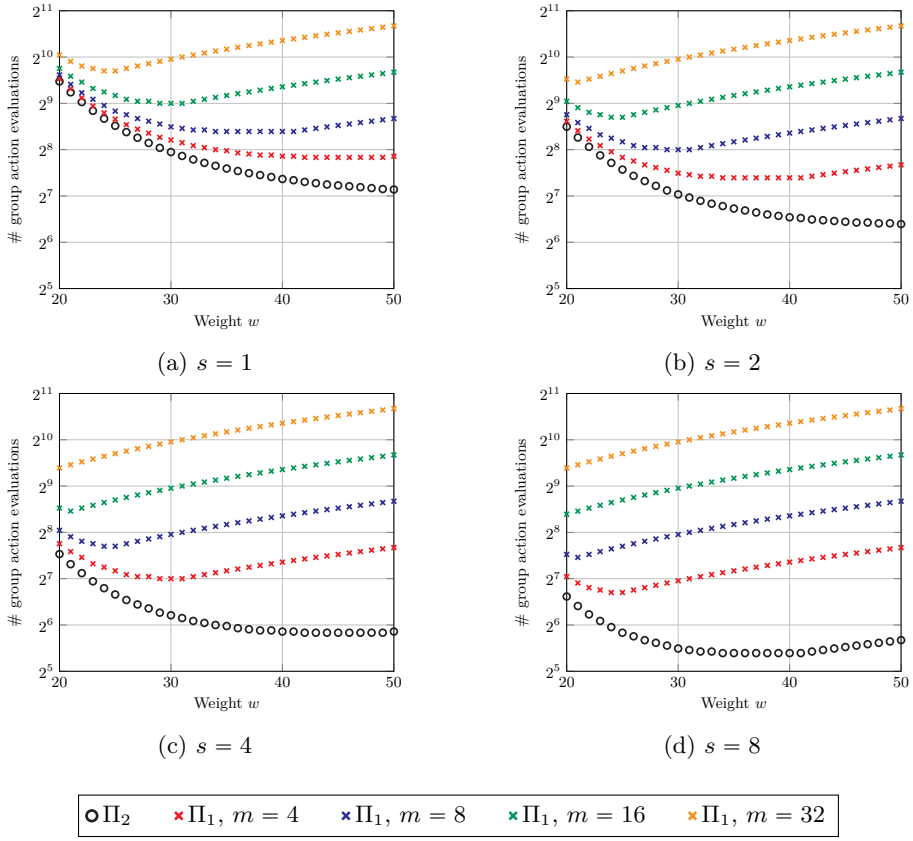


Fig. 12: Number of group action evaluations for $\mathsf{FS}(\Pi_1)$ and $\mathsf{FS}(\Pi_2)$, for $w_1 = w_2 = w$ and $\lambda = 128$.

Removing Fixed-Weight from Π_1. We now tackle the case where protocol Π_1 uses just MPC-in-the-Head with m parties, Multiple Keys with s_1 public keys and t_1 rounds to reach the required λ-bit security, that is, with no assumptions on the weight of $\mathsf{ch}^{(1)} \in \Pi_1.\mathsf{ChSet}$. In this case, the equivalence with a protocol Π_2 using just Fixed-Weight Challenges, Multiple Keys with s_2 public keys and t_2 rounds is not guaranteed. In fact, when we represent a challenge vector $\mathsf{ch}^{(1)} \in \Pi_1.\mathsf{ChSet}_1$ as a challenge $\mathsf{ch}^{(2)}$ for Π_2, we are not assured that there is fixed number of $\mathsf{ch}^{(1)}_j = m$; instead, each one is sampled uniformly from a set of size $sm + 1$. However, $\mathsf{ch}^{(2)}$ still contains enough structure, as we know that each chunk of size t_1 has weight at most 1 and the weight of $\mathsf{ch}^{(2)}$ is therefore upper bounded by t_1. It is therefore natural to analyse the gap between such a protocol Π_1 and a protocol Π_2 using Fixed-Weight Challenges with $t_2 = t_1 m$ rounds and weight $w_2 = t_1$. The following theorem shows that when the number of rounds t_1 for Π is large enough, then Π_2 has the same computational complexity and almost the same response size, but a larger challenge space.

Theorem 3. *Let Π_1 be a protocol obtained using a combination of Multiple Keys and MPC-in-the-Head, with parameters s_1, $m \geqslant 2$ and t_1 as usual. Let Π_2 be a protocol obtained using Multiple Keys and Fixed-Weight Challenges, with $s_2 = s_1$ public values in the public key, $t_2 = t_1 m$ rounds of which $w_2 = t_1$ rounds with non-zero challenge $\mathsf{ch}_j^{(2)} \neq 0$. Then, the difference between the response size of Π_2 and the average response size for Π_1 is $\frac{t_1(\ell_G - \lambda)}{s_1 m + 1}$. Moreover, the two protocols have the same computational complexity and, for any number of keys, $m \geqslant 2$ and $t \geqslant 7$, protocol Π_2 has a larger challenge space than Π_1.*

The proof is given in Appendix D.

Remark 7. Because of the bounds we have used within the proof, the condition $t_1 \geqslant 7$ is a bit loose. For instance, with a single public key and $m = 2$, the initial inequality is already satisfied by any $t_1 \geqslant 5$. Moreover, we have conservatively considered only the case of $s_1 = 1$, but when s_1 increases the inequality starts being satisfied by lower values of t_1. For instance, for $s_1 = 2$ and $m = 2$, $t_1 \geqslant 3$ is enough to guarantee that $|\Pi_2.\mathsf{ChSet}| > |\Pi_1.\mathsf{ChSet}|$.

6.2 MPC-in-the-Head with Puncturable PRFs

The previous section does not take into account the use of a puncturable PRF, such as Seed Tree, to communicate the values $\tilde{g}_j$. Still, taking into account such a puncturable PRF does not change the overall picture. We will now briefly give some motivations, considering the seed tree technique and, for simplicity, the approximation $w \log_2(t/w)$ for the number of released nodes.

For the protocol Π_1 from Theorem 3 with $w_1 = w$ and t_1 rounds combined with Seed Tree, we get a response size of

$$\begin{aligned} |\Pi_1.\mathsf{rsp}^{(2)}| &= w \log\left(\frac{t_1}{w}\right)\lambda + w\big(\log(m)\lambda + \ell_G\big) \\ &= w \log\left(\frac{t_1 m}{w}\right)\lambda + w\ell_G \end{aligned}$$

Now, taking Π_2 with $w_2 = w$ and $t_2 = t_1 m$ and using Seed Tree with the same approximation, we get the very same signature size, as we release $w \log(t_1 m/w)$ nodes of size λ for the $(t_2 - w)$ zero rounds, and w group elements of size ℓ_G for the non-zero rounds. Hence, considerations analogous to the ones in previous section hold: the resulting signature schemes $\mathsf{FS}(\Pi_1)$ and $\mathsf{FS}(\Pi_2)$ would either have the same signature size, with $\mathsf{FS}(\Pi_1)$ requiring more group actions computations, or, the same computational complexity, with $\mathsf{FS}(\Pi_1)$ requiring larger values of w_1, hence, larger signatures.

6.3 Faster Verification using Skipping Edges with Fixed-Weight

This section compares the verification cost of Skipping Edges with Fixed-Weight Challenges, compared to only using Fixed-Weight Challenges. Although Theorems 2 and 3 show that Fixed-Weight Challenges almost always outperforms MPC-in-the-Head in response size, computation cost, and challenge space, this no longer holds when we consider Skipping Edges. We show that the verification cost of Skipping Edges combined with Fixed-Weight Challenges can outperform a protocol using only Fixed-Weight Challenges. To do this, we compare the number of group action evaluations in verification, where we vary the value of the weight w, and increase t accordingly to reach λ-bit security for different configurations of the following protocols.

1. The protocol Π_1 using Fixed-Weight Challenges with t rounds and weight w, combined with Seed Tree, and Multiple Keys using s public keys (Section 5.1),
2. The protocol Π_2 using Skipping Edges with m parties, Fixed-Weight Challenges with t rounds and weight w, Seed Tree, and Multiple Keys with s public keys (Section 5.3).

For Π_1, the number of group action evaluations in signing and verification, which we denote by n_1, are equal, whereas for Π_2 there is a significant difference in the number of group action evaluations between signing, which we denote by n_2^{sign}, and verification, which we denote by n_2^{verif}. Figure 13 shows n_1, n_2^{sign} and n_2^{verif} for different values of w, s and m.

As Figure 13 shows, the smaller w is, the larger the difference between n_2^{verif} and n_1 is, while keeping n_2^{sign} reasonable. This happens because a small weight w requires a larger number of rounds t which reduces the ratio η from (40). When we furthermore consider the number of public keys s for Multiple Keys, we see that for some values s, m and w, we may even get $n_2^{\text{verif}} > n_1$, hence no improvement in verification cost. This happens because the number of rounds t decreases with larger s hence making Skipping Edges less effective overall.

In practice, w only gives a heuristic comparison for the signature sizes of Π_1 and Π_2, as it influences the largest part of signature size whenever $\ell_G \gg \lambda$. In a real-use scenario, in particular scenarios with efficient representations of group elements and hence ℓ_G only slightly larger than λ, the size of the seed and commitment strings has to be taken into account carefully. This is shown in Section 7, where we apply such a precise analysis to existing protocols.

7 Analysis of Signature Schemes based on Group Actions

This section analyses some instantiations of the protocols based on group actions, using the language and results laid out in the previous sections. Specifically, we give a high-level overview of the candidates of the additional call for signatures issued by NIST based on group actions, namely LESS [4], MEDS [18], and ALTEQ [11], using the unified terminology developed in this work.

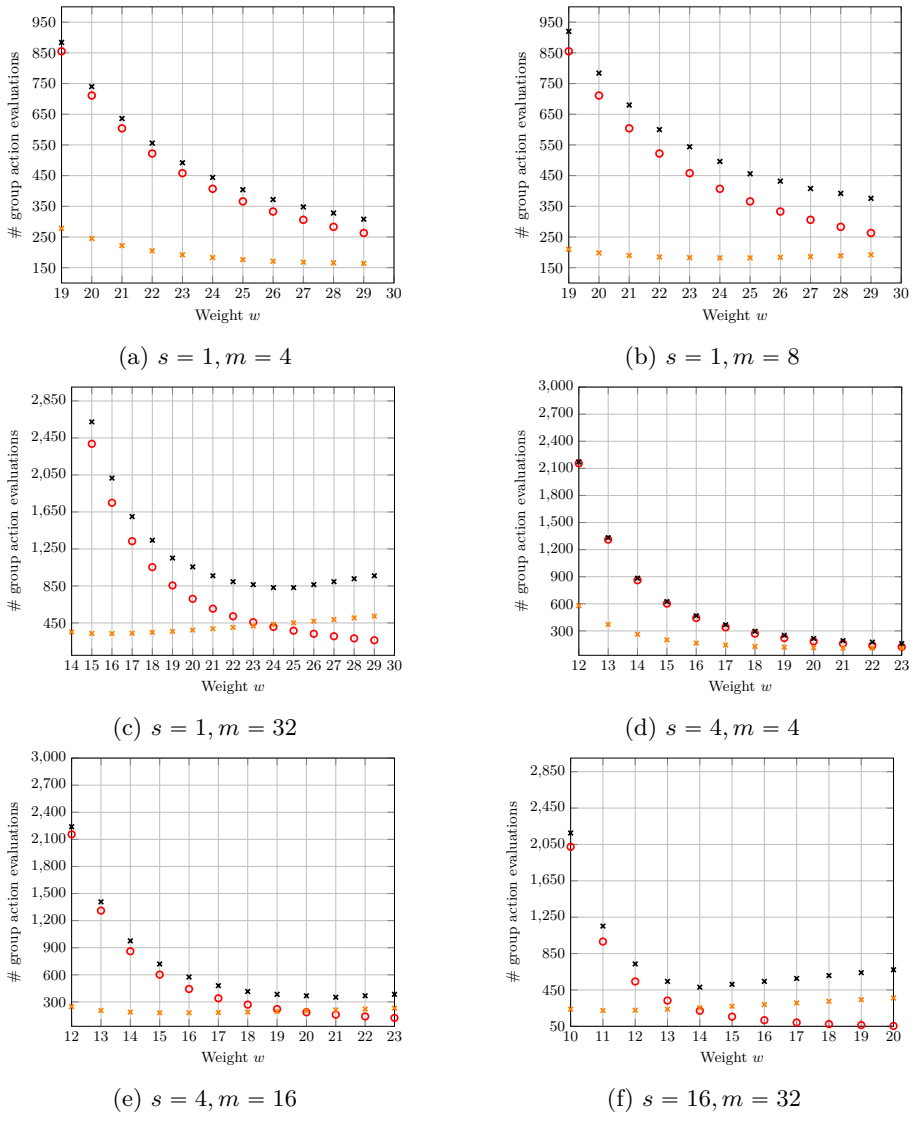


Fig. 13: Comparison between the number of group action evaluations for Π_1, using only Fixed-Weight Challenges, and Π_2, using Fixed-Weight Challenges and Skipping Edges, while varying the weight w.

So far, we have only considered group actions as a black box. In this section, we dive into specific group actions and explain how they gain efficiency on two levels: first, using the generic techniques for group-action based Σ-protocols as described in Sections 3 to 5, and second, using techniques which are specific to each group action, such as more compact representations of group elements or set elements.

7.1 LESS

LESS [4, 6] is a signature scheme based on the hardness of the *Linear (code) Equivalence Problem* (LEP). At its core, linear code equivalence can be seen as a group action: the group G of *monomial maps* acts on the set of k-dimensional linear codes of length n over the finite $\mathbb{F}_q$, with $k \geqslant n$ positive integers. These monomial maps are isometries (i.e. weight preserving) for the Hamming metric and consist of a permutation combined with non-zero scaling factors. Thus, G can be seen as the composition of a permutation matrix with a diagonal matrix of non-zero entries, that is, $G = S_n \times (\mathbb{F}_q^*)^n$.

Hardness Assumption. The linear code equivalence problem asks, given two $[n,k]_q$ linear codes $\mathcal{C}, \mathcal{C}'$, to find a monomial map μ such that $\mathcal{C}' = \mu(\mathcal{C})$. Let $\mathbf{G}$ denote a generator matrix for $\mathcal{C}$, and similarly $\mathbf{G}'$ for $\mathcal{C}'$, then the problem can be re-framed as finding the monomial matrix $\mathbf{Q}$ such that $\mathbf{G}'$ and $\mathbf{GQ}$ have the same systematic form, i.e. generate the same linear subspace. For NIST Security Category I, the LESS specification document uses the parameters

$$q = 127, \;\; n = 252, \;\; k = 126.$$

Size of Elements. A random monomial map in $S_n \times (\mathbb{F}_q^*)^n$ can be represented in $\ell_G = n \cdot (\log n + \lceil \log q \rceil)$ bits. A random $[n,k]$ linear code over $\mathbb{F}_q$ in systematic form can be represented by a $k \times n$ matrix in $\mathbb{F}_q$ containing the identity matrix of dimension k as a submatrix, so $k(n-k)\lceil \log q \rceil$ bits are enough to represent it.

Scheme Optimisation. The LESS scheme instantiates Protocol 2 with the group action described above, and applies the transformations Fixed-Weight Challenges with Seed Tree, as well as Multiple Keys. The current NIST Security Category I specification proposes three sets of values:

- *balanced* has $t = 247$ rounds, fixed-weight $w = 30$, and $s+1 = 2$ public keys;
- *intermediate* has $t = 244$ rounds, fixed-weight $w = 20$, and $s + 1 = 4$ public keys;
- *short* has $t = 198$ rounds, fixed-weight $w = 17$, and $s + 1 = 8$ public keys.

Note that, in [4], the authors use s to indicate the total number of linear codes used as public keys, included the randomly-generated one.

The MPC-in-the-Head and Skipping Edges techniques are not used; indeed, these would not improve the signature size performance of the scheme, as detailed in Section 6. Note that the group action is not commutative, hence Hypercube would not apply.

Specific LESS Improvements. To improve the performance of LESS beyond the above description, Chou, Persichetti, and Santini [20] introduce the concept of *canonical forms*, which allow to compress the information contained in the monomial map $\mathbf{Q}$. While more information on this can be found in [4, 20], here we only briefly point out the most relevant results:

- if two linear codes can be mapped to equivalent codes with the same canonical forms, then the equivalence between the two codes can be found in polynomial time;
- the equivalence can be verified up to canonical form by using much less information than the one contained in $\mathbf{Q}$ (the exact compression depends on the choice of the canonical form).

In this way, the communication cost required to verify the equivalence is substantially reduced, for example from 473 bytes to 237 bytes using a weaker form of canonical forms (see [4, 36]) to just 32 bytes in [20], using the most well-known choice for a canonical form. However, this also implies that the compressed monomial maps cannot be composed anymore, since this way $\ell_G \approx 2\lambda$. Thus, using this optimisation limits us to paths of length 1 during verification, making it impossible to combine this optimisation with MPC-in-the-Head and in particular with Skipping Edges.

Concrete Values. For NIST Security Category I, the specification submitted to round 1 [4] yields

$$\ell_G = 237 \text{ bytes}, \quad \ell_X = 13,892 \text{ bytes}, \quad \lambda = 128,$$

and therefore sizes for the public key and signature are

$$\begin{aligned} \mathsf{pk} &= 13,7 \text{ Kbytes}, \quad \mathsf{sig} = 8,4 \text{ Kbytes} \quad \text{(balanced)}, \\ \mathsf{pk} &= 41,1 \text{ Kbytes}, \quad \mathsf{sig} = 6,1 \text{ Kbytes} \quad \text{(intermediate)}, \\ \mathsf{pk} &= 95,9 \text{ Kbytes}, \quad \mathsf{sig} = 5,2 \text{ Kbytes} \quad \text{(short)}. \end{aligned}$$

7.2 MEDS

MEDS [18, 19] is a signature scheme based on *Matrix Code Equivalence* (MCE). At its core, matrix code equivalence can be seen as the action of the *group* $G = \mathrm{GL}_n(q) \times \mathrm{GL}_m(q)$ on the *set* of k-dimensional matrix codes over $\mathbb{F}_q$. The group elements $(\mathbf{A}, \mathbf{B}) \in \mathrm{GL}_n(q) \times \mathrm{GL}_m(q)$ are isometries and map a code $\mathcal{C}$, given by a basis $\mathbf{C}_1, \ldots, \mathbf{C}_k \in \mathbb{F}_q^{n \times m}$, to an equivalent code $\mathcal{D}$ given by $\mathbf{D}_i = \mathbf{A}\mathbf{C}_i\mathbf{B}$.

Hardness Assumption. The matrix code equivalence problem asks to find $(\mathbf{A}, \mathbf{B})$ given the two (scrambled) bases $(\mathbf{C}_1, \ldots, \mathbf{C}_k)$ and $(\mathbf{D}_1, \ldots, \mathbf{D}_k)$ for the k-dimensional matrix codes $\mathcal{C}$ and $\mathcal{D}$ such that $\mathbf{A} \cdot \mathcal{C} \cdot \mathbf{B} = \mathcal{D}$, i.e. that there exists an invertible matrix $\mathbf{T} \in \mathrm{GL}_k(q)$ such that for all $i = 1, ..., k$:

$$\mathbf{D}_i = \sum_{j=1}^{k} t_{i,j}\ \mathbf{A} \cdot \mathbf{C}_j \cdot \mathbf{B}\ .$$

For NIST Security Category I, the MEDS specifications [18] use the parameters

$$q = 4093,\ n = 14,\ m = 14,\ k = 14.$$

Size of Elements. A random isometry $(\mathbf{A}, \mathbf{B}) \in \mathrm{GL}_n(q) \times \mathrm{GL}_m(q)$ can be represented in $\ell_G = (n^2 + m^2) \cdot \lceil \log q \rceil$ bits. A random k-dimensional matrix code $\mathcal{C} \subseteq \mathbb{F}_q^{n \times m}$ can be represented by k matrices $\mathbf{C}_i \in \mathbb{F}_q^{n \times m}$. By vectorising these matrices, one gets a generator matrix $\mathbf{G}$ of size $k \times mn$, which can be represented in standard form using $\ell_X = k \cdot (n \cdot m - k) \lceil \log q \rceil$ bits. For the NIST Security Category I parameters above we would get $\ell_G = 588$ bytes and $\ell_X = 3822$ bytes.

Scheme Optimisation. The MEDS scheme instantiates Protocol 2 with the group action described above, and applies the transformations Fixed-Weight Challenges with Seed Tree, and Multiple Keys. The current NIST Security Category I specification proposes two sets of values:

- set 1 has $t = 1152$ rounds, fixed-weight $w = 14$, and $s + 1 = 4$ public keys;
- set 2 has $t = 192$ rounds, fixed-weight $w = 20$, and $s + 1 = 5$ public keys.

For both sets MPC-in-the-Head is not used, and this aligns with our arguments presented in Section 6.1. Furthermore, the group action is not commutative, hence Hypercube does not apply. However, as ℓ_G is quite a bit larger than λ, Skipping Edges combined with Fixed-Weight Challenges, Multiple Keys and Seed Tree has the potential to improve on both parameter sets for some aspects of the scheme. We readily compute

- set 3 with $t = 129$ rounds, fixed-weight $w = 10$, $m = 87$ parties and $s = 3$ public keys.

Specific MEDS Improvements. Two techniques improve the performance of MEDS beyond the description above, using techniques specifically tailored for the MCE group action.

1. Section 3 of [18] details how to reduce the public key size, i.e. set elements, to $k - 2$ matrices instead of k, reducing ℓ_X to $(k-2) \cdot (nm - k) \cdot \lceil \log q \rceil$ bits.
2. Section 8 of [18] details how to reduce the isometry representation size, e.g. group elements, using only two matrices $\mathbf{C}, \mathbf{C}' \in \mathbb{F}_q^{n \times m}$. Both can be defined in terms of the (known) basis $(\mathbf{C}_i, \ldots, \mathbf{C}_k)$ as $\mathbf{C} = \sum \lambda_i \mathbf{C}_i$ with $\lambda_i \in \mathbb{F}_q$, hence, both $\mathbf{C}$ and $\mathbf{C}'$ can be represented in $k \cdot \lceil \log q \rceil$ bits. This reduces ℓ_G to $2k \cdot \lceil \log q \rceil$.

Concrete Values. For NIST Security Category I, the MEDS specifications do not apply the second optimalization yet. To give a fair comparison, we stick with the numbers as given in the specification. This gives us

$$\ell_G = 588 \text{ bytes}, \quad \ell_X = 3,297 \text{ bytes}, \quad \lambda = 128,$$

and therefore sizes for the public key and signature as

$$\begin{aligned} \mathsf{pk} &= 9,923 \text{ bytes}, \quad \mathsf{sig} = 9,896 \text{ bytes} \quad \text{(set 1)}, \\ \mathsf{pk} &= 13,220 \text{ bytes}, \quad \mathsf{sig} = 12,976 \text{ bytes} \quad \text{(set 2)}, \\ \mathsf{pk} &= 9,923 \text{ bytes}, \quad \mathsf{sig} = 9,192 \text{ bytes} \quad \text{(set 3, new)}. \end{aligned}$$

We see that the third set of parameters decreases the signature size and requires roughly only half the average group action evaluations in verification (569 against 1152). Signing, however, is negatively impacted, and becomes about 10 times slower. If we include the above optimisation, ℓ_G decreases substantially to 42 bytes and the ratio with λ drops to only 2.6×. This makes Skipping Edges much less effective, and we find that set 3 does not improve anymore.

7.3 ALTEQ

ALTEQ [11, 38] is a signature scheme based on the hardness of the *Alternate Trilinear Form Equivalence* (ATFE) problem. At its core, the alternate trilinear form equivalence can be seen as a group action, where the *group* $G = \mathrm{GL}_n(q)$ acts on the *set* of alternating trilinear forms

$$\varphi : \mathbb{F}_q^n \times \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q.$$

Group elements $\mathbf{A} \in \mathrm{GL}_n(q)$ map φ to $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \varphi(\mathbf{Ax}, \mathbf{Ay}, \mathbf{Az})$.

Hardness Assumption. The alternate trilinear form equivalence problem asks to find $\mathbf{A} \in \mathrm{GL}_n(q)$ given two equivalent trilinear forms φ and ψ. For NIST Security Category I, the ALTEQ specification document uses the parameters

$$q = 2^{32} - 5, \ n = 13.$$

Size of Elements. A random $\mathbf{A} \in \mathrm{GL}_n(q)$ can be represented in $\ell_G = n^2 \cdot \lceil \log q \rceil$ bits. A random alternating trilinear form φ can be described as a linear combination of $e_i^* \wedge e_j^* \wedge e_k^*$, where $1 \leqslant i < j < k \leqslant n$, which form a (known) basis of the space of alternating trilinear forms. The form φ can thus be represented using $\binom{n}{3}$ coefficients in $\mathbb{F}_q$, giving $\ell_X = \binom{n}{3} \cdot \lceil \log q \rceil$.

Scheme Optimisation. The ALTEQ scheme instantiates Protocol 2 with the group action described above, and applies the transformations Fixed-Weight Challenges and Multiple Keys. The current NIST Security Category I specification proposes two sets of values:

– set 1 has $t = 84$ rounds, fixed-weight $w = 22$, and $s = \quad 7$ public keys;
– set 2 has $t = 16$ rounds, fixed-weight $w = 14$, and $s = 458$ public keys.

The technique MPC-in-the-Head is not applied in the specification. Altough in [11, Remark 1.4], the authors propose to analyse MPC-in-the-Head to improve the scheme, the results in Section 6 show this will not improve the performance of ALTEQ. Note that the group action is not commutative, hence Hypercube does not apply.

The current submission of ALTEQ does not use Seed Tree, although in [11, Remark 1.4] the authors write that more understanding is required. Using Proposition 2, we can quantify the gain of using Seed Tree as 272 bytes for set 1 parameters while for set 2 there is no improvement.

Currently, the representation of group elements in ALTEQ is a major limitation. For example, the current NIST Security Category I parameters have $\ell_G \approx 40 \cdot \lambda$. Hence, we can use the combination of Skipping Edges and Fixed-Weight Challenges (Section 5.3) to improve the signature size while controlling the verification time. We readily compute

– set 3 with $t = 164$ rounds, fixed-weight $w = 10$, $m = 29$ parties and $s = 7$ public keys.

To find this set, we capped the average number of group action evaluations during verification to 4×84 (for this set we need 314), so that we expect verification to be 4 times slower for set 3 parameters than for set 1 parameters. Signature generation requires 4756 group action evaluations, making it roughly 60 times slower than set 1. However, the parameter set allows us to reach a signature size below $10,000$ Kbytes while still keeping the size of the public key reasonable. We stress that, when using only Fixed-Weight Challenges and not Skipping Edges, we require at least 1234 rounds to get such small signatures.

Concrete Values. For NIST Security Category I, this gives us

$$\ell_G = 676 \text{ bytes}, \quad \ell_X = 1144 \text{ bytes}, \quad \lambda = 128,$$

and therefore the following sizes for public keys and signatures:

$$\begin{aligned}
\mathsf{pk} = \quad & 8,024 \text{ bytes}, \quad \mathsf{sig} = 15,640 \text{ bytes} \quad \text{(set 1, with Seed Tree)}, \\
\mathsf{pk} = 523,968 & \text{ bytes}, \quad \mathsf{sig} = \quad 9,528 \text{ bytes} \quad \text{(set 2)}, \\
\mathsf{pk} = \quad & 8,024 \text{ bytes}, \quad \mathsf{sig} = \quad 9,992 \text{ bytes} \quad \text{(set 3, new)}.
\end{aligned}$$

References

[1] https://csrc.nist.gov/Projects/Post-Quantum-Cryptography. 2017.

[2] C. Aguilar-Melchor, N. Gama, J. Howe, A. Hülsing, D. Joseph, and D. Yue. "The return of the SDitH". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2023, pp. 564–596.

[3] N. Alamati, L. De Feo, H. Montgomery, and S. Patranabis. "Cryptographic Group Actions and Applications". In: *ASIACRYPT*. Springer. 2020, pp. 411–439.

[4] M. Baldi, A. Barenghi, L. Beckwith, J.-F. Biasse, A. Esser, K. Gaj, K. Mohajerani, G. Pelosi, E. Persichetti, M.-J. O. Saarinen, P. Santini, and R. Wallace. *Matrix Equivalence Digital Signature*. https://www.less-project.com/LESS-2023-08-18.pdf. Accessed: 2023-09-15. 2023.

[5] A. Barenghi, J.-F. Biasse, E. Persichetti, and P. Santini. "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.

[6] A. Barenghi, J.-F. Biasse, E. Persichetti, and P. Santini. "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.

[7] W. Beullens, S. Katsumata, and F. Pintore. "Calamari and Falafl: Logarithmic (Linkable) Ring Signatures from Isogenies and Lattices". In: *Advances in Cryptology – ASIACRYPT 2020*. Ed. by S. Moriai and H. Wang. Cham: Springer International Publishing, 2020, pp. 464–492. ISBN: 978-3-030-64834-3.

[8] W. Beullens, T. Kleinjung, and F. Vercauteren. "CSI-FiSh: efficient isogeny based signatures through class group computations". In: *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I*. Springer. 2019, pp. 227–247.

[9] J.-F. Biasse, G. Micheli, E. Persichetti, and P. Santini. "LESS is More: Code-Based Signatures Without Syndromes". In: *AFRICACRYPT*. Ed. by A. Nitaj and A. Youssef. Springer, 2020, pp. 45–65.

[10] M. Bläser, Z. Chen, D. H. Duong, A. Joux, N. T. Nguyen, T. Plantard, Y. Qiao, W. Susilo, and G. Tang. "On digital signatures based on isomorphism problems: QROM security, ring signatures, and applications". In: *Cryptology ePrint Archive* (2022).

[11] M. Blaser, D. H. Duong, A. K. Narayanan, T. Plantard, Y. Qiao, A. Sipasseuth, and G. Tang. *ALTEQ Signature Specification*. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/ALTEQ-Spec-web.pdf. Accessed: 2023-10-11. 2023.

[12] D. Boneh, J. Guan, and M. Zhandry. "A Lower Bound on the Length of Signatures Based on Group Actions and Generic Isogenies". 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. 507–531.

[13] D. Boneh, J. Guan, and M. Zhandry. "A Lower Bound on the Length of Signatures Based on Group Actions and Generic Isogenies". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2023, pp. 507–531.

[14] D. Boneh and B. Waters. "Constrained Pseudorandom Functions and Their Applications". In: *Advances in Cryptology - ASIACRYPT 2013*. Ed. by K. Sako and P. Sarkar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 280–300. ISBN: 978-3-642-42045-0.

[15] G. Brassard and M. Yung. "One-way group actions". In: *Conference on the Theory and Application of Cryptography*. Springer. 1990, pp. 94–107.

[16] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *Advances in Cryptology – ASIACRYPT 2018*. Springer. 2018, pp. 395–427.

[17] A. Chailloux. "On the (In) security of optimized Stern-like signature schemes". In: WCC. 2022.

[18] T. Chou, R. Niederhagen, E. Persichetti, L. Ran, T. H. Randrianarisoa, K. Reijnders, S. Samardjiska, and M. Trimoska. *Matrix Equivalence Digital Signature*. https://meds-pqc.org/spec/MEDS-2023-05-31.pdf. Accessed: 2023-09-12. 2023.

[19] T. Chou, R. Niederhagen, E. Persichetti, T. H. Randrianarisoa, K. Reijnders, S. Samardjiska, and M. Trimoska. "Take your MEDS: Digital Signatures from Matrix Code Equivalence". In: *AFRICACRYPT* (2023).

[20] T. Chou, E. Persichetti, and P. Santini. *On Linear Equivalence, Canonical Forms, and Digital Signatures*. Cryptology ePrint Archive, Paper 2023/1533. https://eprint.iacr.org/2023/1533. 2023. URL: \url{https://eprint.iacr.org/2023/1533}.

[21] L. De Feo and S. Galbraith. "SeaSign: Compact Isogeny Signatures from Class Group Actions". In: *EUROCRYPT 2019*. Ed. by Y. Ishai and V. Rijmen. Vol. 11478. Lecture Notes in Computer Science. Springer, 2019, pp. 759–789. DOI: 10.1007/978-3-030-17659-4_26. URL: https://doi.org/10.1007/978-3-030-17659-4_26.

[22] L. De Feo and S. D. Galbraith. "SeaSign: compact isogeny signatures from class group actions". In: *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*. Springer. 2019, pp. 759–789.

[23] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[24] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehl é. "Crystals-dilithium: A lattice-based digital signature scheme".

In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), pp. 238–268.

[25] L. Ducas and W. van Woerden. "On the lattice isomorphism problem, quadratic forms, remarkable lattices, and cryptography". In: *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part III*. Springer. 2022, pp. 643–673.

[26] 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.

[27] 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.

[28] A. Fiat and A. Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology — CRYPTO' 86*. Ed. by A. M. Odlyzko. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.

[29] 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.

[30] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. "Zero-knowledge from secure multiparty computation". In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. 2007, pp. 21–30.

[31] 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.

[32] D. Kales and G. Zaverucha. "Improving the performance of the picnic signature scheme". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), pp. 154–188.

[33] J. Katz, V. Kolesnikov, and X. Wang. "Improved non-interactive zero knowledge with applications to post-quantum signatures". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 525–537.

[34] E. Kiltz, D. Masny, and J. Pan. "Optimal security proofs for signatures from identification schemes". In: *In Annual International Cryptology Conference*. Springer. 2016, pp. 33–61.

[35] 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.

[36] E. Persichetti and P. Santini. "A New Formulation of the Linear Equivalence Problem and Shorter LESS Signatures". In: *Cryptology ePrint Archive* (2023).

[37] A. Stolbunov. "Cryptographic schemes based on isogenies". In: (2012).

[38] G. Tang, D. H. Duong, A. Joux, T. Plantard, Y. Qiao, and W. Susilo. "Practical post-quantum signature schemes from isomorphism problems of trilinear forms". In: *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part III*. Springer. 2022, pp. 582–612.

A Action Graphs Protocols

Protocol 2 is a very elementary construction which is used as a building block in the construction of many digital signature schemes. The introduction of Section 3 describes *action graphs* as a framework to capture the transformations presented in Sections 3 and 4. This appendix describes such action graphs in more generality and in particular describes Σ-protocols in much broader generality, so that every transformed Σ-protocol can be seen as an instance of such a general Σ-protocol. Furthermore, we prove security of such a Σ-protocol.

Recall that for a group action $\star : G \times X \to X$ on a finite set, $\mathcal{O}(x)$ denotes the orbit of $x \in X$ under the action of G, that is

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

and that if the action of G is free then $\mathcal{O}(x)$ contains $|G|$ elements. If $\star$ is also transitive, then $\mathcal{O}(x) = X$. We define the action graph $\mathcal{G}_X$ as the graph with nodes X and an edge between $x_1, x_2 \in X$ if there is a $g \in G$ such that $x_2 = g \star x_1$. When considering only the orbit $\mathcal{O}(x)$ of an $x \in X$, we denote the corresponding action graph by $\mathcal{G}_x$. As the action is transitive on the orbit, $\mathcal{G}_x$ is connected, as any pair of vertices (x_1, x_2) is connected by an edge g such that $x_2 = g \star x_1$. When $\star$ is free on $\mathcal{O}x$, this g is moreover unique.

The 1-bit scheme Group Action Σ-Protocol can be interpreted as a subgraph of $\mathcal{G}_x$, where we perform a length-one path starting from x to get $\tilde{x}$. The secret path is $y = g \star x$ for a secret $g \in G$ and we reveal either the edge $(x, \tilde{x})$ or $(y, \tilde{x})$ in the protocol. We may wonder what happens if we consider more general subgraphs of $\mathcal{G}_x$ or we reveal different paths on it. A similar approach was used by Boneh, Guan, and Zhandry [13] to prove a lower bound on the size of signatures based on group actions. However, the authors only use subgraphs for a theoretical bound on the size of the transcript, and do not consider other optimisations such as puncturable PRFs to generate random group elements.

A.1 A Generalized Σ-Protocol for Action Subgraphs

For a generalized Σ-protocol, we sample random group elements $\tilde{g}_1, \tilde{g}_2, ...,$and compute the path on the edges of $\mathcal{G}_x$ by

$$x \mapsto \tilde{g}_1 \tilde{x}_1 \mapsto \tilde{g}_2 \ldots$$

Additionally, we can consider other paths starting from x or other intermediate nodes $\tilde{x}_i$. The public keys are represented by the particular nodes $y_i = g_i \star x$, where the $g_i \in G$ are the associated secret keys. In this way, we obtain a directed subgraph of $\mathcal{G}_x$ that we can use for an identification protocol to prove knowledge on the secret paths g_i. The intuitive idea behind this construction is that, assuming $\lambda \ll \ell_G$, the randomly-generated paths $x \to \tilde{x}_i$ on the subgraph can be revealed efficiently using methods such as Seed Tree or other puncturable PRFs. The following protocol describes the identification protocol in detail.

Protocol 3 (Action Subgraph Σ-Protocol) *Consider an origin $x \in X$, then a **public key** is another node $y_i \in X$ with associated **secret key** a walk $g_i \in G$ such that $g_i \star x = y_i$. Consider s public keys $y_1, ..., y_s$ and a positive integer t representing the number of rounds. Relabel the origin x as y_0. Then the generalized Σ-protocol on $\mathcal{G}_x$ is given as follows.*

1. *a **commitment** is composed by a list of nodes $\tilde{x}_j \in X$ for $j = 1, ..., t$ not equal to any public key y_i. To construct $\tilde{x}_j$, the prover performs paths of length one given by randomly generated $\tilde{g}_j \in G$, starting from the origin x or other nodes previously reached by other walks.*
2. *a **challenge** is a map $c : [1, t] \to [0, s]$, where s is the number of public keys, that encodes a request for all the links $y_{c(j)}$ to $\tilde{x}_j$ for $j \in [1, t]$. The challenge is sampled randomly from a predefined challenge set.*
3. *a **response** for a challenge c is a list of $g'_j \in G$ for $j \in [1, t]$ where each g'_j gives a path $\tilde{x}_j \xrightarrow{g'_j} y_c(j)$.*
4. *the verifier checks that all recomputed nodes corresponds to the committed ones.*

The above protocol generalizes most transformations in Sections 3 and 4. By giving a single proof of its security properties, we give an alternative proof to the security of each transformation in the main body.

Proposition 3. *Protocol 3 is complete, honest-verifier zero-knowledge, and 2-special sound for the mGAIP problem.*

Proof. We prove each property one by one.
Completeness. Since each node $\tilde{x}_j$ is sampled via a random walk from the origin x the Prover can link it to $y_{c(j)}$ using the knowledge of the secret key $g_{c(j)}$, i.e. by the group action properties he can always render:

$$y_{c(j)} \xleftarrow{g_{c(j)}} x \xrightarrow{\tilde{g}_*} \cdots \xrightarrow{\tilde{g}_*} \tilde{x}_j \text{ to } y_{c(j)} \xrightarrow{\tilde{g}_* \cdots \tilde{g}_* \cdot g_{c(j)}^{-1}} \tilde{x}_j \ .$$

Special honest-verifier zero knowledge. Since the random walk during the commitment phase is sampled uniformly the elements in the response are always uniformly random on G, in fact for any j:

– when $c(j) = 0$ the response is a product or a combination of uniformly random group elements;
– when $c(j) \neq 0$ the response is a product of the secret key by at least one uniformly random group element, eventually combined with other uniformly random group elements.

So, given any challenge map $c : [1, t] \to 0, M]$, we simulate an indistinguishable transcripts by executing the verification procedure feeding as input random group elements. This way the output corresponds to the commitment while the input random group elements are the simulated response.

2-Special soundness. Suppose the prover can answer two different challenges map $c, c' : [1, t] \to [1, M]$ on the same commitment. Let j be such that $c(j) \neq c'(j)$, so from the response we retrieve $r, r' \in G$ such that $\tilde{x}_j = r \star y_{c(j)} = r' \star y_{c'(j)}$, by group action properties this implies that $y_{c(j)} = (r^{-1}r') \star y_{c'(j)}$, so we have extracted a solution to mGAIP.

B Results for Seed Tree

This section derives the upper bounds used in on the number of nodes required to release the right leaves in a seed tree. We start from following well-known result [7, 31].

Proposition 4. *When t is a power of 2, for any $\mathsf{ch} \in \{0,1\}^t$ with Hamming weight $w = 1$, the size of the output produced by* ReleaseSeeds *is $\log(t)$. More generally, for any $t > 1$ the output size is bounded by $\lceil \log(t) \rceil$.*

Propositions 5 to 7 show that the size of an output of ReleaseSeeds on input a challenge with weight $w > 1$ is always smaller than w times the case of weight 1, by first considering the case where t is a power of 2, and then generalizing to any t, using the binary expansion of t.

Proposition 5. *Let w be any positive integer, and $t = 2^{\lceil \log(w) \rceil}$, that is, the smallest power of 2 such that $t \geqslant w$. Then, for any $\mathsf{ch} \in \{0,1\}^t$ with Hamming weight w, the size of an output of* ReleaseSeeds *is not greater than $w \log(t/w)$.*

Proof. The output size of ReleaseSeeds cannot exceed $t - w$. Let $\lceil \log(w) \rceil = \log(w) + \gamma$, where $0 \leqslant \gamma < 1$. Since by hypothesis $t = 2^{\lceil \log(w) \rceil}$, we get $t = 2^{\log(w)+\gamma} = w2^{\gamma}$ and $t - w = w(2^{\gamma} - 1)$. Now

$$t - w \leqslant w\gamma = w \log(t/w)$$

as $2^{\gamma} - 1 \leqslant \gamma$ holds for $0 \leqslant \gamma < 1$. □

We now show that $w \log(t/w)$ is an upper bound for the output size of ReleaseSeeds when t is a power of 2.

Proposition 6. *Let t be an integer power of 2. Then, for any $\mathsf{ch} \in \{0,1\}^t$ with Hamming weight $w \leqslant t$, the output of* ReleaseSeeds *is not greater than $w \log(t/w)$.*

Proof. Let t be a power of 2, $t \geqslant w$, and $\mathrm{WC}_t(w)$ denote the biggest number of released seeds when the input challenge has Hamming weight w. For $w = 1$ the theorem follows from Proposition 4. For $w \geqslant 2$ we prove the statement, i.e. $\mathrm{WC}_t(w) \leqslant w \log(t/w)$, by an inductive procedure. In particular, we show below that, if $\mathrm{WC}_x(\widetilde{w}) \leqslant \widetilde{w} \log(x/\widetilde{w})$ for every $\widetilde{w} \leqslant w$ and every $x \in \left\{2^{\lceil \log(w) \rceil}, 2 \cdot 2^{\lceil \log(w) \rceil}, 4 \cdot 2^{\lceil \log(w) \rceil} \cdots, t/2\right\}$, then $\mathrm{WC}_t(w) \leqslant w \log(t/w)$. To prove this, we split the seed tree into two subtrees:

- the tree $\mathsf{SubTree}_1$ formed by all nodes that are ancestors of $\underline{\mathsf{leaf}}_1, \cdots, \underline{\mathsf{leaf}}_{t/2}$, i.e., the tree whose root is $\mathsf{seed}_{2,1}$;
- the tree $\mathsf{SubTree}_2$ formed by all nodes that are ancestors of $\underline{\mathsf{leaf}}_{t/2+1}, \cdots, \underline{\mathsf{leaf}}_t$, i.e., the tree whose root is $\mathsf{seed}_{2,2}$.

An example of these subtrees is depicted in Figure 14.

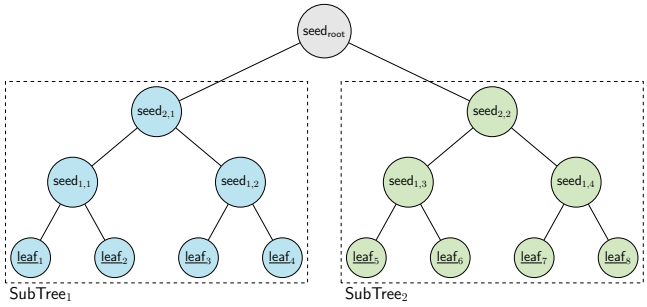


Fig. 14: Example of $\mathsf{SubTree}_1$ and $\mathsf{SubTree}_2$ for $t = 8$.

Given a generic pair (w, t), let $J \subseteq \{1, \cdots, t\}$, of size w, be the set of indices corresponding to the set bits of ch, i.e., for any $i \in J$, we have $\mathsf{ch}_i = 1$. There are three possible cases, which we analyse in the following; for each of them, we prove that $w \log(t/w)$ is an upper for $\mathrm{WC}_t(w)$ under the inductive hypothesis.

a) *Case* $J \subseteq \{1, \cdots, t/2\}$: the function outputs $\mathsf{seed}_{2,2}$ (that is, the root of $\mathsf{SubTree}_2$), plus some seeds from $\mathsf{SubTree}_1$. Note that, if $w = t/2$, then no seed is released from $\mathsf{SubTree}_1$, thus the number of output seeds is $1 \leqslant \frac{t}{2} \log\left(\frac{t}{t/2}\right) = \frac{t}{2}$. Instead, if $w < t/2$, then we get a worst case given by $\mathrm{WC}_{t/2}(w) + 1$. By the inductive hypothesis, we have

$$\mathrm{WC}_{t/2}(w) \leqslant w \log\left(\frac{t}{2w}\right) = w \log(t/w) - w.$$

Hence

$$\mathrm{WC}_{t/2}(w) + 1 \leqslant w \log(t/w) - w + 1 = w \log(t/w) - (w - 1) \leqslant w \log(t/w).$$

b) *Case* $J \subseteq \{t/2 + 1, \cdots, t\}$: analogous to the previous one.

c) *Case* $1 \leqslant |J \cap \{t/2+1, \cdots, t\}| = y < w$: in this case, we have y leaves coming from $\mathsf{SubTree}_1$ and $w-y$ leaves coming from $\mathsf{SubTree}_2$. Hence the worst case is obtained as

$$\mathrm{WC}_t(w) = \max_{y \in \{1,\cdots,w-1\}} \left\{ \mathrm{WC}_{t/2}(y) + \mathrm{WC}_{t/2}(w-y) \right\}.$$

Again, if the inductive hypothesis holds, then $\mathrm{WC}_{t/2}(y) \leqslant y \log\left(\frac{t}{2y}\right)$ and $\mathrm{WC}_{t/2}(w-y) \leqslant (w-y)\log\left(\frac{t}{2(w-y)}\right)$; thus

$$\mathrm{WC}_{t/2}(y) + \mathrm{WC}_{t/2}(w-y) \leqslant y \log\left(\frac{t}{2y}\right) + (w-y)\log\left(\frac{t}{2(w-y)}\right).$$

We now show that, for any $y \in \{1, \cdots, w-1\}$, the above quantity is not greater than $w \log(t/w)$. Indeed, consider that

$$\begin{aligned}
& y \log\left(\frac{t}{2y}\right) + (w-y)\log\left(\frac{t}{2(w-y)}\right) - w\log\left(\frac{t}{w}\right) \\
&= y \log\left(\frac{1}{2y}\right) + (w-y)\log\left(\frac{1}{2(w-y)}\right) - w\log\left(\frac{1}{w}\right) \\
&= y \log\left(\frac{w}{2y}\right) + (w-y)\log\left(\frac{w}{2(w-y)}\right) \\
&= -y \log\left(\frac{y}{w}\right) - (w-y)\log\left(\frac{w-y}{w}\right) - w \\
&= w \cdot \left(-\frac{y}{w}\log\left(\frac{y}{w}\right) - \left(1-\frac{y}{w}\right)\log\left(1-\frac{y}{w}\right) - 1 \right) \\
&= w \cdot \left(h\left(\frac{y}{w}\right) - 1 \right),
\end{aligned}$$

which is always $\leqslant 0$ since $h\left(\frac{y}{w}\right)$, which is the binary entropy function computed on $\frac{y}{w}$, is bounded above by 1.

Before concluding the proof, we discuss the initial case and how to use the inductive step. Let $w = 2$. If $t = 4$, then cases a) and b) leads to the desired inequality, since $w = t/2$. In case c), we have only the possibility $y = 1$, thus we only need $\mathrm{WC}_{t/2}(1) = \mathrm{WC}_2(1) \leqslant 1 \cdot \log(2/1) = 1$. Again, this is guaranteed by Proposition 4, since $\mathrm{WC}_2(1) = 1$. For $t \geqslant 4$ we reason by induction. Namely, knowing that the bound holds for $w = 2$ and $t/2$, we get that it holds also for t thanks to what we discussed in case a), b) and c). All the other values of w are treated similarly. Namely, we start from the fact that the theorem holds for every $w' < w$ and every t and for w and the smallest power of 2 such that $t \geqslant w$ (as seen in Proposition 5). □

We now describe how to deal with the case of t not being an integer power of 2. We start with the easy case of t being the sum of two powers of 2, then extend the proof (again, by induction) on the general case.

Proposition 7. *Let $t = t_1 + t_2$, with t_1 and t_2 being integers powers of 2, $t_1 > t_2$. Then, for any $\mathsf{ch} \in \{0,1\}^t$ with Hamming weight $w \leqslant t$, the output of* ReleaseSeeds *is not greater than $w \log(t/w) + 1$.*

Proof. Let us first consider the case in which all the ones in ch are referred to the subtree with t_1 leaves. In such a case, we just have to release $\mathsf{seed}_{2,2}$ plus the seeds for the subtree of size t_1. Using the same notation as in the proof for Proposition 6, we have that the worst case number of seeds is

$$1 + \mathrm{WC}_{t_1}(w) \leqslant 1 + w \log(t_1/w).$$

Note that this value can be greater than $w \log(t/w)$, since

$$1 + w \log(t_1/w) > w \log(t/w) \implies w < \frac{1}{\log(1/\gamma)}.$$

Note that this inequality is not unrealistic: for instance, it is satisfied by $\gamma = 0.8$ and $w = 2$. However, we show that it is never greater than $1 + \log(t/w)$. Indeed

$$1 + w \log(t_1/w) \leqslant 1 + w \log(t/w) \implies w \log(\gamma) \leqslant 0,$$

which is always true since $\log(\gamma)$ is negative.

We now consider the case in which the w ones in ch are referred to both trees, and show that the resulting worst case is never greater than $w \log(t/w)$. Let $x \in \{1, \cdots, w\}$ denote the number of ones which fall in the right tree, having $t_2 = (1-\gamma)t$ leaves. Then

$$\max_{x \in \{1, \cdots, \min\{w, t_2\}\}} \left\{ \mathrm{WC}_{t_1}(w-x) + \mathrm{WC}_{t_2}(x) \right\}.$$

We now show that, for any x, this is not greater than $w \log(t/w)$. Indeed,

$$\begin{aligned}
&\mathrm{WC}_{t_1}(w-x) + \mathrm{WC}_{t_2}(x) - w \log(t/w) \\
&\leqslant (w-x) \log\left(\frac{t_1}{w-x}\right) + x \log\left(\frac{t_2}{x}\right) - w \log\left(\frac{t}{w}\right). \\
&= (w-x) \log\left(\frac{\gamma t}{w-x}\right) + x \log\left(\frac{(1-\gamma)t}{x}\right) - w \log\left(\frac{t}{w}\right) \\
&= (w-x) \log\left(\frac{\gamma w}{w-x}\right) + x \log\left(\frac{(1-\gamma)w}{x}\right).
\end{aligned}$$

The above quantity is $\leqslant 0$ if

$$1 \geqslant \left(\frac{\gamma w}{w-x}\right)^{w-x} \cdot \left(\frac{(1-\gamma)w}{w-x}\right)^{x} = \frac{(1-\gamma)^x \gamma^{w-x}}{\left(\frac{x}{w}\right)^x \left(\frac{w-x}{w}\right)^{w-x}} = \frac{\widetilde{\gamma}^x (1-\widetilde{\gamma})^{w-x}}{\left(\frac{x}{w}\right)^x \left(1-\frac{x}{w}\right)^{w-x}}.$$

Considering the numerator as a function $f(\widetilde{\gamma})$, we see that it has a maximum in $\widetilde{\gamma} = \frac{x}{w}$. [6] However, the denominator is exactly $f(x/w)$, thus the numerator is never greater than the denominator and the above inequality is satisfied. □

[6] Indeed, its derivative is $\widetilde{\gamma}^{x-1}(1-\widetilde{\gamma})^{w-x-1}\left(x(1-\widetilde{\gamma}) - (w-x)\widetilde{\gamma}\right)$, which has three roots: $\widetilde{\gamma} = 0$, $\widetilde{\gamma} = 1$ and $\widetilde{\gamma} = x/w$. Since $f(\widetilde{\gamma})$ is positive for $\widetilde{\gamma} \in [0;1]$ and $f(0) = f(1) = 0$, $\widetilde{\gamma} = x/w$ corresponds to a maximum.

We are now ready to prove Proposition 2.

Proof. The proof is an immediate consequence of Proposition 7, and can be provided, again, with an inductive procedure. Namely, we start from the hypothesis that $\mathrm{WC}_{\widetilde{t}}(w) \leqslant w\log(t/w) + u - 2$, whenever $\widetilde{t} = \widetilde{t}_1 + \cdots \widetilde{t}_{u-1}$, and show that this implies that for any $t = t_1 + \cdots + t_u$, then $\mathrm{WC}_{\widetilde{t}}(w) \leqslant w\log(t/w) + u - 1$. Since the cases $u = 1$ and $u = 2$ has been treated in Propositions 2 and 6, this is enough to derive all cases by induction. To this end, let us split again the tree into two substructures: the left one, with t_1 leaves, and the right one with $\widetilde{t} = t_2 + \cdots + t_u$ leaves. We bound the worst case as

$$\mathrm{WC}_t(w) \leqslant 1 + \max_{x\in\{1,\cdots,w\}} \{\mathrm{WC}_{t_1}(w-x) + \mathrm{WC}_{\widetilde{t}}(x)\}\,.$$

Since t_1 is a power of 2, from Proposition 6 we get $\mathrm{WC}_{t_1}(w-x) \leqslant (w-x)\log\left(\frac{t_1}{w-x}\right)$, while $\mathrm{WC}_{\widetilde{t}}(x) \leqslant u - 2 + x\log(\widetilde{t}/x)$ from the inductive hypothesis.

$$\begin{aligned} 1 + \mathrm{WC}_{t_1}(w-x) + \mathrm{WC}_{\widetilde{t}}(x) &\leqslant 1 + (w-x)\log\left(\frac{t_1}{w-x}\right) + u - 2 + x\log\left(\frac{\widetilde{t}}{x}\right) \\ &= u - 1 + (w-x)\log\left(\frac{t_1}{w-x}\right) + x\log\left(\frac{\widetilde{t}}{x}\right). \end{aligned}$$

This is not greater than $u - 1 + w\log(t/w)$ if

$$(w-x)\log\left(\frac{t_1}{w-x}\right) + x\log\left(\frac{\widetilde{t}}{x}\right) \leqslant w\log(t/w).$$

We already proved this type of inequality for the proof of Proposition 7, using $t_1 = \gamma t$ and $\widetilde{t} = (1-\gamma)t$, so the inequality holds and the theorem follows. □

C Protocols

This section contains a precise description of the protocols introduced in Section 4 and Section 5.3. It is immediate to transform these protocols into digital signatures using Fiat-Shamir from the given algorithmic description.

The precise protocol of MPC-in-the-Head is as follows.

Protocol 4 (MPC-in-the-Head Group Action Σ-Protocol)

1. *On input $g, y = g \star x$, $m \geqslant 1$ the prover:*
   - *samples m random group elements $\tilde{g}_i \xleftarrow{\$} G$;*
   - *starting from $\tilde{x}_0 = x$ computes sequentially $\tilde{x}_i \leftarrow \tilde{g}_i \star \tilde{x}_{i-1}$ for $i = 1$;*
   - *computes and forward to the verifier the commitment* $\mathsf{cmt} \leftarrow \mathsf{cmt}(\tilde{x}_1, ..., \tilde{x}_m)$.

2. *The verifier samples uniformly at random a challenge* $\mathsf{ch} \in \{0, ..., m\}$*, and sends it to the prover.*
3. *Given* ch*, the prover compute the response composed by* m *group elements as follows:*
   - *if* $\mathsf{ch} \neq m$ *sets* $g'_{\mathsf{ch}+1} \leftarrow \tilde{g}_{\mathsf{ch}+1} \cdots \tilde{g}_1 \cdot g^{-1}$*;*
   - *for* $i = 1, ..., m$ *with* $i \neq \mathsf{ch} + 1$ *sets* $g'_i \leftarrow \tilde{g}_i$*;*
   - *forward to the verifier the responses* $\mathsf{rsp} \leftarrow (\tilde{g}'_1, ..., \tilde{g}'_m)$*.*
4. *The verifies to check* $\mathsf{cmt}, \mathsf{ch}, \mathsf{rsp}$*:*
   - *computes* $\tilde{x}_0 = x$ *and* $\tilde{x}_i \leftarrow g'_i \star \tilde{x}_{i-1}$ *for* $i = 1, ..., \mathsf{ch}$*;*
   - *if* $\mathsf{ch} \neq m$ *sets* $\tilde{x}_{\mathsf{ch}+1} \leftarrow g'_{\mathsf{ch}+1} \star y$*;*
   - *computes* $\tilde{x}_i \leftarrow g'_i \star \tilde{x}_{i-1}$ *for* $i > \mathsf{ch} + 1$*;*
   - *verifies* $\mathsf{cmt} = \mathsf{cmt}_X(\tilde{x}_1, ..., \tilde{x}_m)$ *and outputs 1 (pass) or 0 (fail) accordingly.*

The following protocol describe instead the *left case* for Skipping Edges. The similiarity with Protocol 4 is clear.

Protocol 5 (Skipping edges Σ-Protocol)

1. *On input* $g, y = g \star x$*,* $m \geqslant 1$ *the prover:*
   - *samples* m *random group elements* $\tilde{g}_i \xleftarrow{\$} G$*;*
   - *starting from* $\tilde{x}_0 = x$ *computes sequentially* $\tilde{x}_i \leftarrow \tilde{g}_i \star \tilde{x}_{i-1}$ *for* $i = 1, ..., m$*;*
   - *set* $\mathsf{cmt}_1 = Com(\tilde{x}_1)$ *(we are reversing Remark 5);*
   - *set* $\mathsf{cmt}_j \leftarrow \mathsf{C}(\tilde{x}_j \| \mathsf{cmt}_{j+1})$ *for* $j = 2, ..., m$*;*
   - *computes and forward to the verifier the commitment* $\mathsf{cmt} \leftarrow \mathsf{cmt}_m$*.*
2. *The verifier samples uniformly at random a challenge* $\mathsf{ch} \in \{0, ..., m\}$*, and sends it to the prover.*
3. *Given* ch*, the prover compute the response composed by* m *group elements as follows:*
   - *if* $\mathsf{ch} \neq m$ *sets* $g'_{\mathsf{ch}+1} \leftarrow \tilde{g}_{\mathsf{ch}+1} \cdots \tilde{g}_1 \cdot g^{-1}$*;*
   - *for* $i = 1, ..., m$ *with* $i \neq \mathsf{ch} + 1$ *sets* $g'_i \leftarrow \tilde{g}_i$*;*
   - *forward to the verifier the responses* $\mathsf{rsp} \leftarrow (\tilde{g}'_1, ..., \tilde{g}'_m)$ *and* $\mathsf{cmt}_{\mathsf{ch}-1}$ *when* $\mathsf{ch} > 1$*.*
4. *The verifies to check* $\mathsf{cmt}, \mathsf{ch}, \mathsf{rsp}$*:*
   - *computes* $\tilde{x}'_{\mathsf{ch}} \leftarrow (\tilde{g}'_{\mathsf{ch}} \cdots \tilde{g}'_i) \star x$*;*
   - *if* $\mathsf{ch} \neq m$ *sets* $\tilde{x}'_{\mathsf{ch}+1} \leftarrow g'_{\mathsf{ch}+1} \star y$*;*
   - *computes* $\tilde{x}'_i \leftarrow g'_i \star \tilde{x}'_{i-1}$ *for* $i > \mathsf{ch} + 1$*;*
   - *if* $\mathsf{ch} = 1$ *computes* $\mathsf{cmt}'_1 = \mathsf{C}(\tilde{x}'_1)$*;*
   - *computes* $\mathsf{cmt}'_i \leftarrow \mathsf{C}(\tilde{x}'_i \| \mathsf{cmt}_{i-1})$ *for* $i = \mathsf{ch}, ..., m$*;*
   - *verifies* $\mathsf{cmt} = \mathsf{cmt}'_m$*.*

   Finally, it outputs 1 (pass) or 0 (fail) accordingly.

Hypercube We will now delve into the detailed explanation of the Hypercube protocol outlined previously. Note that since we are assuming commutativity for the group action we usa an additive notation for the group operation. Also, here $S_0(j)$ denotes the integers from 0 to $m-1$ with j-th bit equal to 0, while $S_1(j)$ the one with j-th bit equal to 1. Assume that n is such that $m+1 = 2^n$.

The protocol can then be instantiated as follows.

Protocol 6 (Hypercube MPC-in-the-Head)

1. *On input* $g, y = g \star x$, $n, m \geqslant 1$ *the prover:*
   - *samples* m *random group elements* $\tilde{g}_i \xleftarrow{\$} G$ *for* $i = 0, ..., m-1$*;*
   - *define* $\tilde{g}_m = g - (\tilde{g}_{m-1} + \cdots + \tilde{g}_0)$*;*
   - *for all* $j = 0, ..., n-1$ *compute*
   $$x \xrightarrow{\sum_{i \in S_0(j)} \tilde{g}_i} \tilde{x}_j \ ;$$
   - *computes and forward to the verifier the commitment* $\mathsf{cmt} \leftarrow \mathsf{cmt}(\tilde{x}_1, ..., \tilde{x}_n)$.
2. *The verifier samples uniformly at random a challenge* $\mathsf{ch} \in \{0,1\}^n$*, and sends it to the prover.*
3. *Given* ch*, the prover compute the response composed by* m *group elements as follows:*
   - *computes* i_{ch} *using the negation*[7] *of the binary digits of* ch*, i.e.*
   $$i_{\mathsf{ch}} = (\neg \mathsf{ch}_0) + (\neg \mathsf{ch}_1)2 + \cdots + (\neg \mathsf{ch}_{n-1})2^{n-1} \ ,$$
   - *sets as* rsp *all group elements* $\tilde{g}_i$ *for* $i = 0, ..., m$ *with* $i \neq i_{\mathsf{ch}}$*;*
   - *forward* rsp *to the verifier.*
4. *The verifier parse* ch *as* $\mathsf{ch}_0, ..., \mathsf{ch}_{n-1}$ *and for all* $j = 0, ..., n-1$ *do:*
   - *if* $\mathsf{ch}_j = 0$ *computes:*
   $$x \xrightarrow{\sum_{i \in S_0(j)} \tilde{g}_i} \tilde{x}'_j \ ;$$
   - *if* $\mathsf{ch}_j = 1$ *computes:*
   $$y \xrightarrow{-\tilde{g}_m - \sum_{i \in S_1(d)} \tilde{g}_i} \tilde{x}'_j \ .$$
5. *verifies* $\mathsf{cmt} \leftarrow \mathsf{cmt}(\tilde{x}'_1, ..., \tilde{x}'_n)$*. Finally it outputs 1 (pass) or 0 (fail) accordingly.*

D Security Proofs

D.1 Reduction from mGAIP to GAIP

Proposition 8. *Given an algorithm to solve mGAIP, that runs in time* T *and succeeds with probability* p*, it is possible to solve GAIP, in time approximately equal to* $T + O(\mathsf{poly}(n))$*, with probability of success equal to* $p/2$.

[7] $\neg \mathsf{ch} = \mathsf{ch} + 1 \mod 2$

Proof. Let $\mathcal{A}$ be an adversary for mGAIP. We show how to construct an adversary $\mathcal{A}'$ that can solve GAIP using $\mathcal{A}$ as a subroutine. From a GAIP instance $(x, y = g \star x)$, $\mathcal{A}'$ samples uniformly at random $g_i^{(0)}, ..., g_i^{(r-1)}$. Then, it computes (in polynomial time) half of the elements starting from x, and half starting from y; wlog, we can imagine that $x_i = g_i^{(i)} \star x$ for $i \in [0; r/2 - 1]$, while $x_i = g_i^{(i)} \star y$ for $i \in [r/2; r - 1]$ are generated from y. Since the new instances are randomly generated, they are indistinguishable from the original one. At this point, $\mathcal{A}'$ runs $\mathcal{A}$ on input $x_0, ..., x_{r-1}$, and outputs, with probability p, a response g^*, j, j' such that $x_{j'} = g^* \star x_j$. Now, if the two indices lie in the two different halves of $[0, r]$, it is possible to use the random group element to get g; for example if $j < r/2 < j'$ then $g = \left(g_i^{(j')}\right)^{-1} \cdot g^* \cdot g_i^{(j)}$. Since this happens with probability $1/2$, we get the result. □

D.2 Security Proofs for the New Protocols

For almost all the constructions in the main body their security can be derived as a corollary of Proposition 3, however since both Skipping Edges and Hypercube cannot be framed as special cases of Protocol 3 we need to prove their security separately.

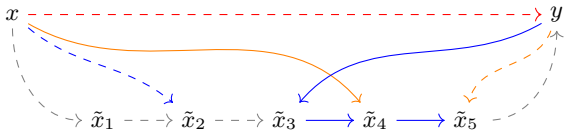
Theorem 4. *For any $n > 0$, the identification protocol Skipping Edges for the GAIP relation $y = g \star x$ is complete, honest-verifier zero knowledge and 2-special sound with soundness error $\frac{1}{m+1}$.*

Proof. The completeness of the protocol is trivial from the description.

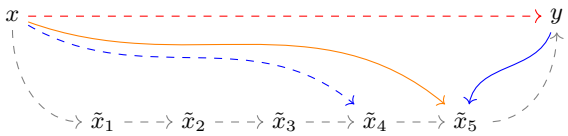
Zero knowledge. To prove zero knowledge, we can use the same simulator as in Proposition 3 to generate the group elements. To generate the commitment values $\{c_l\}_{l<\mathsf{ch}}$, we can simply consider random values on the cmt_X co-domain since we are in the random oracle model.

Special soundness. The 2-special soundness can also be proven with similar strategies as in Proposition 3, with the difference of focusing on the final tail. Let us consider two accepting transcripts with the same commitment, but different challenge values, without loss of generality $\mathsf{ch}_0 < \mathsf{ch}_1$. By the collision resistance of $\mathsf{Merkle.Root}$, we get the same committed value $c_1, ..., c_m$. From the response to ch_1, we get $\tilde{r}$ such that $c_{\mathsf{ch}_1} = \mathsf{cmt}_X(\tilde{r} \star x)$. By combining the responses to ch_0, we have instead $\hat{r}$ such that $c_{\mathsf{ch}_1} = \mathsf{cmt}_X(\hat{r} \star y)$ since $\mathsf{ch}_1 \geqslant \mathsf{ch}_0 + 1$. By the collision resistance of the cmt_X function, we have $\tilde{r} \star x = \hat{r} \star y$ and so $y = (\hat{r}^{-1}\tilde{r}) \star x$, thus we have extracted the secret g proving the special soundness.

Once again, the 2-special soundness immediately implies that the soundness error is the reciprocal of the cardinality of the challenge space, as claimed by the Theorem.



(a) Case $\mathsf{ch}_0 = 2$ in blue and $\mathsf{ch}_1 = 4$ in orange, with $m = 5$.



(b) Case $\mathsf{ch}_0 = m = 5$ in blue and $\mathsf{ch}_1 = 4$ in orange.

Fig. 15: Representation of the action graph available to the knowledge extractor, for two pairs of accepting transcripts. blue is relative to knowledge provided only in transcript ch_0, orange to knowledge provided only in transcript ch_1. Dashed orange and blue edges are known to the extractor, but not used in the extraction.

Example. A graphical representation of the extractor's behavior is given in Figure 15. For instance, for the case $\mathsf{ch}_0 = 2$ and $\mathsf{ch}_1 = 4$, which is reported in the top half of the figure, the extractor can recover the secret g using the following path $x \mapsto \tilde{x}_4 \mapsto \tilde{x}_3 \mapsto y$.

For the case $\mathsf{ch}_0 = 4$, $\mathsf{ch}_1 = 5$, the secret can be extracted from the path $x \mapsto \tilde{x}_5 \mapsto y$.

Note that commitment verification after the interruption plays a crucial role, as this implies that there are common nodes in the paths associated to the two transcripts. Without this, the knowledge extractor would not work. For instance, let us consider again the case depicted in Figure (a) and assume that only the final element $\tilde{x}_5$ is verified. Elements for the transcript ch_1 are denoted with *. In this case, the extractor would know the following four paths

$$x \mapsto \tilde{x}_4, \quad y \mapsto \tilde{x}_5 \ ;$$
$$x \mapsto \tilde{x}_2^*, \quad y \mapsto \tilde{x}_3^* \mapsto \tilde{x}_4^* \mapsto \tilde{x}_5 \ .$$

Notice that only the element $\tilde{x}_5$ is guarantee to be shared, but this would give only information about a circular loop that starts and ends in y. Instead the others, in particular it may be that $\tilde{x}_4 \neq \tilde{x}_4^*$.

D.3 Hypercube Security Proof

Proposition 9. *For any $n > 0$, the identification protocol Hypercube (Protocol 4) for the GAIP relation $y = g \star x$ is complete, honest-verifier zero knowledge and 2-special sound with soundness error 2^{-n}.*

Proof. For the correctness observe that for each j we need all the $\tilde{g}_i$ with j-th digit equal to ch_j, so in particular not i_{ch} since the j-th digit is $\neg\mathsf{ch}_j$, so

the verifier has all the required information to recompute the commitment. The other details are trivial from the protocol description.

Special soundness: Given the response of two different challenges $\mathsf{ch} \neq \mathsf{ch}'$ to the same commitment consider one d with $\mathsf{ch}_d \neq \mathsf{ch}'_d$ and use the same extractor of Proposition 1.

Honest-verifier zero knowledge: To simulate the protocol on a known challenge ch consider the index i_{ch} represented by the negation of the digits of ch, like in the protocol. Then generate uniformly random $\tilde{g}_i \in G$ for $i \neq i_{\mathsf{ch}}$ and repeat the response procedure using them. Then use the resulting set elements as commitment. This clearly lead to a valid transcript. The group elements $\tilde{g}_i \in G$ for $i < m$ are generated as in the protocol, moreover in the honest protocol $\tilde{g}_m$ is uniformly random in G since it is a sum containing elements uniformly random in G, so the transcript is also indistinguishable.

D.4 Proof of Theorem 3

Proof. Protocol Π has challenge space of size size $|\mathsf{ChSet}| = (ms+1)^t$ and the number of computed group actions, which is the same for the verifier and the prover, is equal to mt. The average response size per round is given by

$$\frac{sm}{sm+1}\Big((m-1)\lambda + \ell_G\Big) + \frac{1}{sm+1}\cdot m\lambda = \frac{sm}{sm+1}\left(\lambda\left(m-1+\frac{1}{s}\right)+\ell_G\right).$$

So, considering t parallel executions, one gets an average response size of

$$|\mathsf{rsp}| = \frac{smt}{sm+1}\left(\lambda\left(m-1+\frac{1}{s}\right)+\ell_G\right).$$

We now consider protocol Π'. The computational complexity is given by $t' = mt$, which is the same as that of protocol Π. The response size is instead

$$|\mathsf{rsp}'| = (t'-w')\lambda + w'\ell_G = (mt-t)\lambda + t\ell_G = t(m-1)\lambda + t\ell_G.$$

In particular

$$|\mathsf{rsp}'| - |\mathsf{rsp}| = \frac{t}{sm+1}(\ell_G - \lambda).$$

The challenge space of Π' has size

$$|\mathsf{ChSet}'| = \binom{t'}{w'}s^{w'} = \binom{mt}{t}s^t.$$

We now show that, whenever $t \geqslant 7$, we get $|\mathsf{ChSet}'| > |\mathsf{ChSet}|$. Notice that

$$\binom{mt}{t}s^t > (1+ms)^t \implies \binom{mt}{t} > \left(\frac{1+ms}{s}\right)^t = m^t\left(1+\frac{1}{ms}\right)^t.$$

Since the right-hand term is a decreasing function of s, it is enough to prove the inequality for $s = 1$, that is,

$$\binom{mt}{t} \geqslant m^t \left(1 + \frac{1}{m}\right)^t. \tag{44}$$

Let $h(x) := -x\log(x) - (1-x)\log(x)$ denote the binary entropy function, and consider that

$$\binom{x}{\alpha x} \geqslant \frac{1}{2\sqrt{2x\alpha(1-\alpha)}} \cdot 2^{x \cdot h(\alpha)}, \quad \forall \alpha \in [0;1].$$

In our case, we have $x = mt$ and $\alpha = 1/m$, hence

$$\begin{aligned}
\binom{mt}{t} &\geqslant \frac{1}{2\sqrt{2t(1-1/m)}} \cdot 2^{mt \cdot h(1/m)} \\
&> \frac{1}{2\sqrt{2t}} \cdot 2^{mt \cdot h(1/m)} \\
&= \frac{1}{2\sqrt{2t}} \cdot \left(2^{h(1/m)}\right)^{mt} \\
&= \frac{1}{2\sqrt{2t}} \cdot \left(2^{\frac{1}{m}\log(m) + \left(\frac{m-1}{m}\right)\log\left(\frac{m}{m-1}\right)}\right)^{mt} \\
&= \frac{1}{2\sqrt{2t}} \cdot 2^{t\log(m) + t(m-1)\log\left(\frac{m}{m-1}\right)} \\
&= \frac{1}{2\sqrt{2t}} \cdot m^t \cdot \left(\frac{m}{m-1}\right)^{t(m-1)}.
\end{aligned}$$

So, (44) is satisfied whenever

$$\begin{aligned}
\frac{1}{2\sqrt{2t}} \cdot m^t \left(\frac{m}{m-1}\right)^{t(m-1)} &\geqslant m^t \left(1 + \frac{1}{m}\right)^t \\
\implies \left(\frac{m}{m-1}\right)^{t(m-1)} &\geqslant 2\sqrt{2t} \left(1 + \frac{1}{m}\right)^t.
\end{aligned}$$

With simple manipulations, we rewrite it as

$$\left(\frac{m-1}{m+1}\left(\frac{m}{m-1}\right)^m\right)^t \geqslant 2\sqrt{2t},$$

from which

$$\log\left(\frac{m-1}{m+1}\left(\frac{m}{m-1}\right)^m\right) \geqslant \frac{1}{t}\left(1.5 + \frac{1}{2}\log(t)\right).$$

In the considered range $2 \leqslant m \leqslant \infty$, the left-hand term is an increasing function, hence has minimum value for $m = 2$, equal to $\log(4/3) = 0.4150$. Instead, the right-hand term is a decreasing function of t when $t \geqslant 1$: for $t = 7$, it has value $0.4148 < \log(4/3)$. □