VOLE-in-the-Head Signatures Based on the Linear Code Equivalence Problem

Michele Battagliola[1], Laura Mattiuz[1,3], Alessio Meneghetti[2,3]

[1]Department of Information Engineering, Marche Polytechnic University, Ancona, Italy.
[2]Department of Mathematics, University of Bari Aldo Moro, Bari, Italy.
[3]Department of Mathematics, University of Trento, Trento, Italy.

Contributing authors: battagliola.michele@proton.me; lauramttiuz@gmail.com; alessio.meneghetti@uniba.it;

Abstract

The Vector Oblivious Linear Evaluation in the Head (VOLEitH) paradigm has proven to be a versatile tool to design zero-knowledge proofs and signatures in post-quantum cryptography. In this paper, we propose three VOLE-friendly modellings for Proofs of Knowledge (PoK) of a solution of an instance of the Linear Code Equivalence Problem (LEP). For the first two schemes, we propose two new reductions from LEP to the Multivariate Quadratic (MQ) problem, that may be of independent interest for the cryptanalysis of LEP. Instead, the last model is obtained by generalizing a recent work by Bettaieb et al. to the context of monomial matrices instead of permutation matrices.

While our proposed schemes exhibit larger signature sizes compared to LESS, they improve the computational efficiency, reducing the overall complexity from $\boldsymbol{O(n^3)}$ to $\boldsymbol{O(n^2 \log n)}$ and $\boldsymbol{O(n^2 \log^2 n)}$, where $\boldsymbol{n}$ is the length of the code.

1 Introduction

Following the first call for the Post-Quantum Cryptography (PQC) standardization process started in December 2016, in September 2022 NIST issued a call for additional digital signature schemes [1], with the intent of diversifying the post-quantum signature standards. This stimulated the effort of the cryptographic community to innovate the digital signature scheme landscape, by devising new efficient algorithms and novel approaches for the construction of such protocols. Recently, in October

2024, NIST published the list of the 14 round-2 candidates, which will be further analysed to produce the future standards for post-quantum digital signatures.

Among these 14 proposals, the FAEST signature scheme [2] stands out because of its original construction, namely the *VOLE-in-the-Head* paradigm. *Vector Oblivious Linear Evaluation* (VOLE) has proven to be a useful tool to construct efficient zero-knowledge proofs of knowledge (ZKPoK). VOLE-based proofs use preprocessed random VOLE correlations in a commit-and-prove framework where the VOLE correlations are used to commit to the witness, and then statements are proven about the commitments thanks to homomorphic properties. However, these proofs are inherently designated-verifier, since the verifier must hold its secret part of the VOLE correlation to ensure the soundness. To overcome this drawback, the VOLE-in-the-Head (VOLEitH) paradigm was introduced in [3]: it employs an all-but-one vector commitment scheme to generate VOLE correlations, and allows the construction of publicly verifiable ZK proofs while maintaining the simplicity of VOLE-based protocols. Moreover, combined with the Fiat-Shamir transformation, VOLEitH is a versatile tool to construct efficient signature schemes. The FAEST signature scheme is the first example of VOLEitH signature scheme, and its success in the NIST competition has motivated the cryptographic community to create VOLEitH-friendly protocols for a variety of hard problems in cryptography. For instance, in [4] Bettaieb et al. introduced VOLEitH-friendly modellings for digital signature schemes based on two hard problems in coding theory, namely the Syndrome Decoding Problem and the Permuted Kernel Problem. Moreover, various signature schemes based on MPCitH are now proposing alternative versions based on the new VOLEitH framework, such as Mirath [5], RYDE [6] and SDitH [7].

1.1 Our Contributions

In this work we present three VOLE friendly modellings for the Linear Code Equivalence Problem (LEP), which is at the basis of LESS [8]. In particular, we design threee different VOLE-based protocols, which can be used to create efficient signature schemes using the VOLEitH paradigm together with the Fiat-Shamir heuristic [9]. The first and the second models are based on two distinct reductions from LEP to the Multivariate Quadratic (MQ) problem, and employ an existing VOLE-based protocol targeting this problem from [3] to develop the VOLEitH signatures. To do so, we designed two new reductions from LEP to MQ, that may be of independent interest in the context of solving the Code Equivalence Problem, as they can be exploited for future algebraic attacks such as [10, 11], that are however outside the scope of this paper. Instead, the third approach takes inspiration from [4], and adapts its signature scheme based on the Permuted Kernel Problem (PKP) to the case of LEP.

1.2 Paper organization

In Section 2 we provide the mathematical preliminaries needed for the paper. Section 3 is focused on the VOLE-in-the-Head techniques and presents all the results needed in that field, as well as the VOLE protocol for proving degree-two relations from [3].

Section 4 and Section 5 are the core of our work. In particular, in Section 4 we present our three modellings, while in Section 5 we present the signatures obtained from them and discuss their performances. Finally, in Section 6 we draw our conclusions, and highlight some possible research directions and open problems.

In the appendices Appendix A and Appendix B, we report the pseudocode for the generation of VOLE correlations and the GGM tree construction used for it. Finally, Appendix C contains an alternative description of the model presented in Section 4.3.

2 Preliminaries

In this section we introduce the notation and the various mathematical notions used through this paper. We devote the following section to an extensive explanation of the VOLEitH framework.

2.1 Notations

We fix p to be a prime number and $q = p^r$, $r \geq 1$. We denote by $\mathbb{F}_p$ (respectively, $\mathbb{F}_q$) the finite field with p (respectively q) elements and we write $\mathbb{F}_p^* = \mathbb{F}_p \backslash \{0\}$.

Vectors are denoted by boldface lowercase letters. The vector having all entries equal to 0 is denoted by $\mathbf{0}$: we do not specify its length, as it will always be clear from the context. Similarly, matrices are represented by boldface uppercase letters. In particular, we denote by $\mathbf{I}_n$ the $n \times n$ identity matrix, and by $\mathbf{0}_n$ the $n \times n$ zero matrix. Instead, whenever we consider a rectangular zero matrix, we write $\mathbf{0}_{k,n}$ to indicate that the matrix has k rows and n columns. Let $\mathbf{U}, \mathbf{V}$ be two matrices. Then, we write $[\mathbf{U} \mid \mathbf{V}]$ to denote the matrix obtained by concatenating $\mathbf{U}$ and $\mathbf{V}$ one after the other. Finally, if $\mathbf{v} \in \mathbb{F}_p^n$, we denote by $\mathrm{diag}(\mathbf{v})$ the $n \times n$ diagonal matrix defined by $\mathbf{v}$.

The set of permutations over n elements is denoted by S_n; the action of a permutation over a vector $\mathbf{x} = (x_1, ..., x_n)$ is denoted as $\pi(\mathbf{x}) = (x_{\pi^{-1}(1)}, ..., x_{\pi^{-1}(n)})$. We will often represent a permutation $\pi \in \mathsf{S}_n$ as an $n \times n$ matrix $\mathbf{P}$. With abuse of notation, we also write $\mathbf{P} \in \mathsf{S}_n$. Also, we will make a widespread use of the following sets of matrices over $\mathbb{F}_p$:

- $\mathsf{GL}_k(p)$ stands for the set of invertible $k \times k$ matrices over $\mathbb{F}_p$;
- $\mathsf{M}_n(p)$ denotes the set of monomial matrices with coefficients in $\mathbb{F}_p$, i.e. matrices of the form $\mathbf{Q} = \mathbf{DP}$, where $\mathbf{P}$ is an $n \times n$ permutation matrix and $\mathbf{D} = \mathrm{diag}(\mathbf{v})$ for some $\mathbf{v} \in (\mathbb{F}_p^*)^n$. With abuse of notation, we will often write $\tau \in \mathsf{M}_n(p)$ to indicate that $\tau = (\mathbf{v}, \pi) \in \mathbb{F}_p^{*n} \rtimes \mathsf{S}_n$ is a monomial transformation.

Let $a \leq b$ be two integers: we write $[a, b]$ to represent the set of integers $\{a, a+1, ..., b-1, b\}$. The set $[1, b]$ is simply denoted by $[b]$.

Let S be a set, then $s \xleftarrow{\$} S$ denotes that s is sampled uniformly at random from the set S. If an element s is randomly sampled from S using a seed $\mathtt{sd}$, we will write $s \xleftarrow{\$,\mathtt{sd}} S$.

Finally, let A be an algorithm. If A is deterministic, we write $y \leftarrow \mathsf{A}(X_1, ..., X_n)$ to denote that y is computed as the output of algorithm A with inputs $X_1, ..., X_n$. Instead, if A is a probabilistic algorithm, we use two notations to assign to the variable y the output of A on inputs $X_1, ..., X_n$:

- $y \xleftarrow{\$} \mathsf{A}(X_1, ..., X_n)$ to highlight the probabilistic nature of A;
- $y \leftarrow \mathsf{A}(X_1, ..., X_n; R)$ when the randomness is set and hence the output is uniquely determined.

2.2 Commitment Schemes

A *commitment scheme* is a cryptographic primitive that allows a sender to commit to a chosen value and keep it hidden from other parties. To do so, the sender sends some information derived from his secret value, called *commitment*, to a receiver. Later in the protocol, the sender can choose to *open* his commitment, i.e. reveal the committed value to the receiver by sending a *decommitment*.

Definition 1 (Commitment scheme) A *commitment scheme* is defined by the tuple of algorithms $(\mathsf{Com}, \mathsf{Decom})$, where:

- $(\mathtt{com}, \mathtt{decom}) \xleftarrow{\$} \mathsf{Com}(\mathtt{m})$ is a probabilistic algorithm taking as input a public message $\mathtt{m}$ and computing a commitment $\mathtt{com}$ and a decommitment $\mathtt{decom}$ for the message $\mathtt{m}$;
- $\mathtt{m} \leftarrow \mathsf{Decom}(\mathtt{com}, \mathtt{decom})$ is a deterministic algorithm that, given a commitment $\mathtt{com}$ and a decommitment $\mathtt{decom}$, outputs a message $\mathtt{m}$ if the opening is accepted, or $\perp$ if the opening is rejected.

A commitment scheme must satisfy the following properties:

- *hiding*: the commitment $\mathtt{com}$ does not reveal anything about the original message $\mathtt{m}$;
- *binding*: it must be difficult for the sender to change the committed value once $\mathtt{com}$ is sent.

2.3 Interactive Proofs and Zero-Knowledge Protocols

Intuitively, a *(interactive) zero-knowledge proof* is a two-party interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where the verifier $\mathcal{V}$ learns nothing from the prover $\mathcal{P}$ except for the validity of a statement. To provide a more formal description, we specify the following definitions.

Definition 2 (Binary relation) A *binary relation* is a finite set $R \subseteq X \times Y$, where $X, Y \subseteq \{0,1\}^*$. We say that $y \in Y$ is a *witness* for the *statement* x if $(x, y) \in R$. The set $L_R := \{x \in X \mid \exists y \in Y \text{ s.t. } (x, y) \in R\}$ is called *language* of R.

Definition 3 (Interactive Proof) An *interactive proof* $(\mathcal{P}, \mathcal{V})$ is an interactive protocol between two probabilistic polynomial-time machines $\mathcal{P}$ and $\mathcal{V}$. The *prover* $\mathcal{P}$ takes as input a pair $(x, y) \in R$, while the verifier $\mathcal{V}$ takes as input only the statement x. At the end of the

protocol $\mathcal{V}$ either accepts (outputs 1) or rejects (outputs 0). We denote the output of the protocol with $(\mathcal{P}(y), \mathcal{V})(x)$.

The sequence of messages exchanged in a protocol execution is called *transcript*. We say that a transcript is *accepting* (resp. *rejecting*) if $\mathcal{V}$ accepts (resp. rejects).

In the following, we assume that the prover $\mathcal{P}$ always sends the first and last message within an execution of an interactive proof. Moreover, we refer to an interactive proof having $2\mu + 1$ rounds of communication as a $2\mu + 1$-*round interactive proof*.

Definition 4 (Public Coin) An interactive proof $(\mathcal{P}, \mathcal{V})$ is said to be *public-coin* if all the random choices of the verifier $\mathcal{V}$ are made public.

In a public-coin interactive proof $(\mathcal{P}, \mathcal{V})$, the verifier only needs to send to the prover his random choices. In this context, the messages sent by the verifier are called *challenges*, denoted $\mathtt{chl}_i$. In each round of communication, the verifier samples the challenge $\mathtt{chl}_i$ from a *round challenge set* Ch_i. The structure of a $2\mu+1$-round public coin interactive proof is summarized in Algorithm 1.

Algorithm 1 $2\mu + 1$-round public coin interactive proof $(\mathcal{P}, \mathcal{V})$

$\mathcal{P}(x, y)$		$\mathcal{V}(x)$
$\mathtt{com} \leftarrow \mathcal{P}_0$	$\xrightarrow{\mathtt{com}}$	
	$\xleftarrow{\mathtt{chl}_1}$	$\mathtt{chl}_1 \xleftarrow{\$} \mathsf{Ch}_1$
$\mathtt{rsp}_1 \leftarrow \mathcal{P}_1(y, \mathtt{com}, \mathtt{chl}_1)$	$\xrightarrow{\mathtt{rsp}_1}$	
	$\vdots$	
	$\xleftarrow{\mathtt{chl}_\mu}$	$\mathtt{chl}_\mu \xleftarrow{\$} \mathsf{Ch}_\mu$
$\mathtt{rsp}_\mu \leftarrow \mathcal{P}_\mu(y, \mathtt{com}, \{\mathtt{chl}_j, \mathtt{rsp}_j\}_{j\in[\mu-1]}, \mathtt{chl}_\mu)$	$\xrightarrow{\mathtt{rsp}_\mu}$	**return** $\mathcal{V}(x, \mathtt{com}, \{\mathtt{chl}_j, \mathtt{rsp}_j\}_{j\in[\mu]})$

2.4 Signature Schemes

A *signature scheme* is a public-key cryptographic protocol used to ensure integrity, authenticity and non-repudiation.

Definition 5 (Digital Signature Scheme) A *digital signature scheme* is defined by a tuple of algorithms $\mathsf{DS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ where

– $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(\lambda)$: on input a security parameter λ, it outputs the public parameters;

– $(\mathtt{sk}, \mathtt{pk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathtt{pp}, \lambda)$: KeyGen is a probabilistic *key generation* algorithm that, given as inputs the public parameters $\mathtt{pp}$ and the security parameter λ, outputs a pair of public and private keys $(\mathtt{pk}, \mathtt{sk})$;
– $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathtt{sk}, \mathtt{m})$: Sign is a (usually) probabilistic *signature* algorithm that takes as input a private key sk and a message $\mathtt{m}$, and outputs a signature σ for $\mathtt{m}$;
– $b \leftarrow \mathsf{Verify}(\mathtt{pk}, \mathtt{m}, \sigma)$: Verify is a deterministic algorithm that takes as inputs a public key $\mathtt{pk}$, a message $\mathtt{m}$ and a signature σ, and outputs a bit $b \in \{0, 1\}$, where $b = 1$ means "accept" and $b = 0$ means "reject".

Moreover, it is required that $\Pr\left(\mathsf{Verify}(\mathtt{pk}, \mathtt{m}, \sigma) = 1 \mid \sigma = \mathsf{Sign}(\mathtt{sk}, \mathtt{m}; R)\right) = 1$.

In the following we will omit to explicitly mention the Setup algorithm unless required.

An important notion for the security of signature schemes is that of *unforgeability under chosen message attack*, where an attacker knows many message-signature pairs and tries to produce a *forgery*, that is a new valid message-signature pair that was computed without having access to the secret key.

Definition 6 (Existential unforgeability under chosen message attack) Consider a digital signature scheme $\mathsf{DS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. Let $\mathcal{A}$ be an adversary having access to a signing oracle $\mathcal{O}^{DS}(\cdot)$. Define the following experiment where M denotes the set of the messages queried by $\mathcal{A}$ to $\mathcal{O}^{DS}$.

$\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathsf{DS},\mathcal{A}}(\lambda)$:

1 : $\mathtt{pp} \xleftarrow{\$} \mathsf{Setup}(\lambda)$
2 : $(\mathtt{pk}, \mathtt{sk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathtt{pp})$
3 : $\mathtt{m}||\sigma \xleftarrow{\$} \mathcal{A}^{\mathcal{O}^{\mathsf{DS}}(\cdot)}(\mathtt{pk})$
4 : **if** $\mathtt{m} \in \mathsf{M}$
5 : **return** 0
6 : **else**
7 : **return** $\mathsf{Verify}(\mathtt{pk}, \mathtt{m}, \sigma)$

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}^{\mathsf{euf-cma}}_{\mathsf{DS},\mathcal{A}}(\lambda) = \Pr(\mathsf{Exp}^{\mathsf{euf-cma}}_{\mathsf{DS},\mathcal{A}}(\lambda) = 1)$. DS is existential unforgeable against chosen message attacks if $\mathbf{Adv}^{\mathsf{euf-cma}}_{\mathsf{DS},\mathcal{A}}(\lambda)$ is negligible for every probabilistic polynomial time forger $\mathcal{A}$.

Fiat-Shamir Transformation

Introduced in [9], the Fiat-Shamir transformation is an extensively used technique for constructing digital signature schemes starting from public-coin interactive proofs. Intuitively, the idea behind the transformation is to replace the challenge-communication

step with the evaluation of a hash function. In particular, given a $(2\mu+1)$-interactive proof $(\mathcal{P},\mathcal{V})$ for a hard relation $R \subseteq X \times Y$ the obtained Fiat-Shamir signature is obtained as in Algorithm 2.

Algorithm 2 Fiat-Shamir tranformation of a $(2\mu+1)$-round interactive proof

$\mathsf{KeyGen}(\lambda)$

1 : $(x,y) \xleftarrow{\$} R$
2 : $\mathtt{sk} \leftarrow y$
3 : $\mathtt{pk} \leftarrow x$
4 : **return** $(\mathtt{pk},\mathtt{sk})$

$\mathsf{Sign}(\mathtt{sk},\mathtt{m})$

1 : $\mathtt{com} \leftarrow \mathcal{P}_0(\mathtt{sk})$
2 : **for** $i \in [\mu]$ **do**
3 : $\quad \mathtt{chl}_i \leftarrow \mathsf{H}_i(\mathtt{com}, \{\mathtt{chl}_j, \mathtt{rsp}_j\}_{j\in[i-1]}, \mathtt{m})$
4 : $\quad \mathtt{rsp}_i \leftarrow \mathcal{P}_i(\mathtt{sk}, \mathtt{com}, \{\mathtt{chl}_j, \mathtt{rsp}_j\}_{j\in[i-1]}, \mathtt{chl}_i)$
5 : **return** $\sigma \leftarrow (\mathtt{com}, \mathtt{rsp}_1, ..., \mathtt{rsp}_\mu)$

$\mathsf{Verify}(\mathtt{pk},\mathtt{m},\sigma)$

1 : $(\mathtt{com}, \mathtt{rsp}_1, ..., \mathtt{rsp}_\mu) \leftarrow \sigma$
2 : **for** $i \in [\mu]$ **do**
3 : $\quad \mathtt{chl}_i \leftarrow \mathsf{H}_i(\mathtt{com}, \{\mathtt{chl}_j, \mathtt{rsp}_j\}_{j\in[i-1]}, \mathtt{m})$
4 : **return** $\mathcal{V}(x, \mathtt{com}, \{\mathtt{chl}_j, \mathtt{rsp}_j\}_{j\in[\mu]})$

The work [12], later extended by [13] to cover the general multi-round case, shows the minimal conditions that $(\mathcal{P},\mathcal{V})$ should satisfy so that the obtained signature is EUF-CMA secure.

2.5 Code Equivalence Problem

A $[n,k]$ linear code $\mathcal{C}$ over $\mathbb{F}_q$ is a k-dimensional subspace of $\mathbb{F}_q^n$. We say that n is the length of the code while k is its dimension. An elements $\mathbf{x} \in \mathcal{C}$ is called codeword. A code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with dimension k can be represented through a *generator matrix*, that is, a full-rank $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ such that the rows form a basis for the code. Another representation can be obtained through a basis for the null space of the code, which is a subspace of $\mathbb{F}_q^n$ with dimension $n-k$ called *dual code* and indicated as $\mathcal{C}^\perp$. A basis for $\mathcal{C}^\perp$ is a full-rank matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, called *parity-check matrix*. For any generator matrix $\mathbf{G}$ and parity-check matrix $\mathbf{H}$ for the same code, it holds that $\mathbf{G}\mathbf{H}^\top = \mathbf{0}$. We introduce here the Code Equivalence Problem, which is at the basis of LESS signature.

Definition 7 (Permutation Code Equivalence) Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two $[n,k]$-linear codes over $\mathbb{F}_q$. We say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are *permutation equivalent*, and we write $\mathcal{C}_1 \overset{\mathsf{PE}}{\sim} \mathcal{C}_2$, if there exists a permutation $\pi \in \mathsf{S}_n$ such that

$$\mathcal{C}_2 = \pi(\mathcal{C}_1) = \{\pi(\mathbf{x}) \mid \mathbf{x} \in \mathcal{C}_1\}.$$

The above definition can be rewritten in terms of generator matrices. Let $\mathbf{G}_1$ and $\mathbf{G}_2$ be the $k \times n$ generator matrices of $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. In this context,

we say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are permutation equivalent if there exist an invertible matrix $\mathbf{S} \in \mathsf{GL}_k(q)$ and an $n \times n$ permutation matrix $\mathbf{P}$ such that

$$\mathbf{G}_2 = \mathbf{S}\mathbf{G}_1\mathbf{P}.$$

Notice that the invertible matrix $\mathbf{S}$ is necessary because the representation of the code by the generator matrix is not unique.

We can extend Definition 7 using linear isometries, i.e. $\tau = (\mathbf{v}, \pi) \in \mathbb{F}_q^{*n} \rtimes \mathsf{S}_n$ such that

$$\tau(\mathbf{x}) = (v_1 x_{\pi^{-1}(1)}, ..., v_n x_{\pi^{-1}(n)}),\ \mathbf{x} \in \mathbb{F}_q^n.$$

We obtain the following more general definition.

Definition 8 (Linear Code Equivalence) Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two $[n, k]$-linear codes over $\mathbb{F}_q$. We say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are *linearly equivalent*, and we write $\mathcal{C}_1 \overset{\mathsf{LE}}{\sim} \mathcal{C}_2$ if there exists a linear isometry $\tau = (\mathbf{v}, \pi) \in \mathbb{F}_q^{*n} \rtimes \mathsf{S}_n$ such that

$$\mathcal{C}_2 = \tau(\mathcal{C}_1) = \{\tau(\mathbf{x}) \mid \mathbf{x} \in \mathcal{C}_1\}.$$

Remark 1 Permutation equivalence is a special case of linear equivalence: it suffices to consider the above definition with $\mathbf{v} = (1, ..., 1) \in \mathbb{F}_q^n$.

Similarly to the permutation equivalence, we can reformulate the definition of linear code equivalence in terms of the generator matrices. In this case, the application of a linear isometry corresponds to the multiplication by a monomial matrix $\mathbf{Q}$. Therefore, we say that $\mathcal{C}_1$ and $\mathcal{C}_2$ are linearly equivalent if there exist an invertible matrix $\mathbf{S} \in \mathsf{GL}_k(q)$ and a monomial matrix $\mathbf{Q} \in \mathsf{M}_n(q)$ such that

$$\mathbf{G}_2 = \mathbf{S}\mathbf{G}_1\mathbf{Q}.$$

Notice that given an isometry $\tau = (\mathbf{v}, \pi) \in \mathbb{F}_q^{*n} \rtimes \mathsf{S}_n$ the associated monomial matrix is $\mathbf{Q} = \mathrm{diag}(\mathbf{v})\mathbf{P}$ where $\mathrm{diag}(\mathbf{v})$ denotes the diagonal matrix having the vector $\mathbf{v}$ on the diagonal and $\mathbf{P}$ is the permutation associated to π.

The above definition is the foundation of the following hard problem that is at the basis of LESS.

Problem 1 ((Computational) Linear Equivalence Problem (LEP)) *Let* $\mathbf{G}_1, \mathbf{G}_2 \in \mathbb{F}_q^{k \times n}$ *be the generator matrices of two linearly equivalent codes* $\mathcal{C}_1, \mathcal{C}_2$. *Find* $\mathbf{S} \in \mathsf{GL}_k(q)$ *and* $\mathbf{Q} \in \mathsf{M}_n(q)$ *such that* $\mathbf{G}_2 = \mathbf{S}\mathbf{G}_1\mathbf{Q}$.

For a detailed discussion about the hardness of the Code Equivalence Problem, we refer to [14].

3 VOLE-in-the-Head

In this section, we first recall the VOLE-in-the-head technique introduced in [3] and then we focus on the VOLE-in-the-Head protocol for multivariate quadratic polynomial systems of [3, Section 6].

3.1 VOLE correlations

Vector Oblivious Linear Evaluation (VOLE) [15] is a two-party functionality $\mathcal{F}_{\mathsf{VOLE}}$ that allows two parties to share a chosen linear combination of two chosen vectors. Formally, given a finite field $\mathbb{F}_p$ and positive integers ℓ, r, the VOLE functionality takes two vectors $(\mathbf{u}, \mathbf{v}) \in \mathbb{F}_p^\ell \times \mathbb{F}_{p^r}^\ell$ from a sender, which we call *prover* $\mathcal{P}$, and a scalar $\Delta \in \mathbb{F}_{p^r}$ from a receiver, here called *verifier* $\mathcal{V}$, and outputs to $\mathcal{V}$ the linear combination

$$\mathbf{r} = \mathbf{u} \cdot \Delta + \mathbf{v}. \tag{1}$$

While this is the classical definition of VOLE, most VOLE-based protocols use a randomized version. Here, the VOLE functionality generates correlated randomness, called *VOLE correlation* (or random VOLE correlation), $(\mathbf{u}, \mathbf{v}) \in \mathbb{F}_p^\ell \times \mathbb{F}_{p^r}^\ell$ and $(\Delta, \mathbf{r}) \in \mathbb{F}_{p^r} \times \mathbb{F}_{p^r}^\ell$ such that $\mathbf{r} = \mathbf{u} \cdot \Delta + \mathbf{v}$. Then, $\mathcal{P}$ learns $(\mathbf{u}, \mathbf{v})$, while $\mathcal{V}$ learns $(\Delta, \mathbf{r})$.

VOLE correlations can be used as *linear homomorphic commitments*. Indeed, given a VOLE correlation satisfying Equation (1), the prover $\mathcal{P}$ is committed to the vector $\mathbf{u}$ towards the verifier $\mathcal{V}$. It can be easily seen that VOLE commitments are both hiding and binding:

- **hiding**: the verifier $\mathcal{V}$ does not know the random mask $\mathbf{v}$, so it cannot recover $\mathbf{u}$;
- **binding**: in order to provide an opening to a different value $\mathbf{u}' \neq \mathbf{u}$, the prover needs to guess Δ.

In an ideal scenario, the VOLE correlation would have been generated by a trusted functionality $\mathcal{F}_{\mathsf{VOLE}}$ which would send privately $(\mathbf{u}, \mathbf{v})$ to the prover and $(\Delta, \mathbf{r})$ to the verifier. However, in practice the correlations are generated by one party, who then sends the respective parts to the other participant. For instance, in [16], the author proposes SoftSpokenOT, a novel silent protocol to implement $\mathcal{F}_{\mathsf{VOLE}}$ over polynomial-sized fields $\mathbb{F}_{p^r}$ using a $(N-1)$-out-of-N oblivious transfer (OT). Indeed, let $\mathbf{t}_0, ..., \mathbf{t}_{N-1} \in \mathbb{F}_p^\ell$ be the messages held by $\mathcal{P}$ and let $\Delta \in [0, N-1]$ be an index chosen by $\mathcal{V}$, who then learns the messages $\mathbf{t}_x$ for $x \neq \Delta$. Define $\mathbf{u} \in \mathbb{F}_p^\ell$ and $\mathbf{v} \in \mathbb{F}_{p^r}^\ell$ as follows:

$$\mathbf{u} := \sum_{x \in \mathbb{F}_{p^r}} \mathbf{t}_x \ , \ \mathbf{v} := \sum_{x \in \mathbb{F}_{p^r}} \mathbf{t}_x(-x). \tag{2}$$

Then $\mathcal{V}$ can locally compute $\mathbf{r} \in \mathbb{F}_{p^r}^\ell$ such that $\mathbf{r} = \mathbf{u} \cdot \Delta + \mathbf{v}$ as

$$\mathbf{r} := \sum_{x \in \mathbb{F}_{p^r} \setminus \{\Delta\}} \mathbf{t}_x(\Delta - x) = \sum_{x \in \mathbb{F}_{p^r}} \mathbf{t}_x(\Delta - x) = \mathbf{u} \cdot \Delta + \mathbf{v}. \tag{3}$$

3.2 Subspace VOLE

The protocol introduced in the previous section efficiently generates VOLE correlations only when p^r is small, since the process requires $\mathcal{O}(p^r)$ operations. However, when VOLE correlations are used in ZK proofs, we need Δ to have enough entropy to ensure an adequate knowledge error. While the standard VOLE protocol from [16] does not efficiently achieve this security, a solution is easily found using *subspace VOLE* [16].

Subspace VOLE (sVOLE) is a more general instance of VOLE where the correlated randomness is given by matrices instead of vectors. Namely, the participants of an instance of sVOLE hold $(\mathbf{U}, \mathbf{V})$ and $(\mathbf{\Delta}, \mathbf{R})$ such that $\mathbf{U} \in \mathbb{F}_p^{\ell \times k_\mathcal{C}}$, $\mathbf{V}, \mathbf{R} \in \mathbb{F}_{p^r}^{\ell \times n_\mathcal{C}}$ and

$$\mathbf{R} = \mathbf{U}\mathbf{G}_\mathcal{C} \operatorname{diag}(\mathbf{\Delta}) + \mathbf{V}, \tag{4}$$

where $\mathbf{G}_\mathcal{C} \in \mathbb{F}_p^{k_\mathcal{C} \times n_\mathcal{C}}$ is the the generator matrix of a $[n_\mathcal{C}, k_\mathcal{C}]$-linear code $\mathcal{C}$, and $\mathbf{\Delta} \in \mathbb{F}_{p^r}^{n_\mathcal{C}}$. Essentially, sVOLE consists of $n_\mathcal{C}$ independent VOLE correlations placed side-by-side, except that the rows of $\mathbf{U}^* = \mathbf{U}\mathbf{G}_\mathcal{C}$ are restricted to be codewords of $\mathcal{C}$. More details on general subspace VOLE can be found in [3, §5].

As shown in [17], the idea is to use subspace VOLE to generate an instance of standard VOLE over an exponentially large field to achieve the soundness requirements. To do so, the prover and the verifier first engage in the small-field VOLE protocol τ times, thus generating τ independent VOLE correlations over $\mathbb{F}_{p^r}$. Denote the correlations as $(\mathbf{u}_i, \mathbf{v}_i)$ and $(\Delta_i, \mathbf{r}_i)$ for $i \in [0, \tau - 1]$. By placing these VOLE instances side-by-side, $\mathcal{P}$ and $\mathcal{V}$ form the matrices $\mathbf{U}' \in \mathbb{F}_p^{\ell \times \tau}$ and $\mathbf{V}, \mathbf{R}' \in \mathbb{F}_{p^r}^{\ell \times \tau}$ such that

$$\mathbf{R}' = \mathbf{U}' \cdot \operatorname{diag}(\mathbf{\Delta}) + \mathbf{V} \tag{5}$$

where $\mathbf{\Delta} = (\Delta_1, ..., \Delta_\tau)^T \in \mathbb{F}_{p^r}^\tau$. The prover $\mathcal{P}$ can fix the τ VOLEs to use the same $\mathbf{u}$, say $\mathbf{u} = \mathbf{u}_0$, by computing the corrections $\mathbf{c}_i = \mathbf{u}_i - \mathbf{u}$ for $i \in [\tau - 1]$ and sending them to the verifier $\mathcal{V}$. In this way, $\mathcal{V}$ can correct his own output by computing

$$\mathbf{R} = \mathbf{R}' - [\mathbf{0} \mid \mathbf{C}] \cdot \operatorname{diag}(\mathbf{\Delta}) = [\mathbf{u} \mid \cdots \mid \mathbf{u}] \cdot \operatorname{diag}(\mathbf{\Delta}) + \mathbf{V} \tag{6}$$

where $\mathbf{C} = [\mathbf{c}_1 \mid \cdots \mid \mathbf{c}_{\tau-1}]$. Equation (6) provides an instance of subspace VOLE: this is more evident if we let $\mathcal{C} = \mathcal{C}_{\mathsf{rep}}$ be the repetition code of length τ over $\mathbb{F}_p$ and rewrite Equation (6) as

$$\mathbf{R} = \mathbf{u} \cdot [1 \mid \cdots \mid 1] \cdot \operatorname{diag}(\mathbf{\Delta}) + \mathbf{V} \tag{7}$$

with $[1 \mid \cdots \mid 1]$ being the generator matrix of $\mathcal{C}_{\mathsf{rep}}$. Finally, if we view $\mathbf{\Delta}$ and the rows of $\mathbf{V}$ and $\mathbf{R}$ as elements of $\mathbb{F}_{p^{r\tau}}$, we obtain a single VOLE correlation over $\mathbb{F}_{p^{r\tau}}$ satisfying:

$$\mathbf{r}^* = \mathbf{u} \cdot \Delta^* + \mathbf{v}^*. \tag{8}$$

Let λ denote the security parameter. Since now $\Delta^* \in \mathbb{F}_{p^{r\tau}}$, if we fix $\tau = \left\lceil \frac{\lambda}{r \log_2(p)} \right\rceil$, the probability that $\mathcal{P}$ guesses it is equal to $\frac{1}{p^{r\tau}} \approx \frac{1}{2^\lambda}$.

This procedure can then be used to construct standard VOLE correlations over large fields, which provide a small enough soundness error when used in a ZK proof. However, notice that the prover $\mathcal{P}$ can cheat when sending the corrections $\{\mathbf{c}_i\}_{i\in[\tau-1]}$. If any $\mathbf{c}_i$ is generated incorrectly, $\mathcal{V}$ will not be able to properly correct his share and the ZK proof will not have the same security guarantees. Hence, the two parties need to engage in a *VOLE consistency check*, as explained in [16]. The consistency check is realized applying a random linear universal hash function to $\mathbf{u}$ and $\mathbf{V}$. The linear hash function consists of a compression matrix $\mathbf{H}^{\mathsf{UHF}} \in \mathbb{F}_{p^r}^{m\times\ell}$, which is sampled by $\mathcal{V}$ and sent as a challenge to $\mathcal{P}$, who computes

$$\tilde{\mathbf{u}} = \mathbf{H}^{\mathsf{UHF}}\mathbf{u} \quad , \quad \tilde{\mathbf{V}} = \mathbf{H}^{\mathsf{UHF}}\mathbf{V}. \tag{9}$$

$\mathcal{P}$ then sends $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{V}}$ to $\mathcal{V}$, who computes $\tilde{\mathbf{R}} = \mathbf{H}^{\mathsf{UHF}}\mathbf{R}$ and aborts the protocol if

$$\tilde{\mathbf{R}} \neq \tilde{\mathbf{u}} \cdot [1 \mid \cdots \mid 1] \cdot \mathrm{diag}(\boldsymbol{\Delta}) + \tilde{\mathbf{V}}. \tag{10}$$

To make sure that $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{V}}$ do not reveal any information about the original $\mathbf{u}$ and $\mathbf{V}$, the linear hash function $\mathbf{H}^{\mathsf{UHF}}$ must satisfy the following property.

Definition 9 ($\mathbb{F}_p^\ell$-hiding matrix [16, Definition 4.1]) A matrix $\mathbf{A} \in \mathbb{F}_{p^r}^{m\times(\ell+h)}$ is $\mathbb{F}_p^\ell$-*hiding* if the distribution of $\mathbf{Av}$ is independent of $\mathbf{v}_{[\ell]}$ when $\mathbf{v} \xleftarrow{\$} \mathbb{F}_p^{\ell+h}$.

Hence, $\mathcal{P}$ and $\mathcal{V}$ need to use VOLE correlations of length $\hat{\ell} = (\ell + h)$, whose first ℓ entries are used to commit to the witness, and whose last h entries are needed to perform the consistency check without any leakage. Indeed, by using an $\mathbb{F}_p^\ell$-hiding hash function $\mathbf{H}^{\mathsf{UHF}} \in \mathbb{F}_{p^r}^{m\times(\ell+h)}$, the first ℓ entries of $\mathbf{u}$ will remain hidden to the verifier $\mathcal{V}$. Also, notice that if $\mathbf{H}^{\mathsf{UHF}}$ is $\mathbb{F}_p^\ell$-hiding, then it is also $\mathbb{F}_{p^r}^\ell$-hiding: therefore, the first ℓ rows of $\mathbf{V}$ will also remain hidden to $\mathcal{V}$.

3.3 VOLE-in-the-Head

VOLE-in-the-Head (VOLEitH) is a technique proposed by Baum et al. [3] which exploits the homomorphic properties of VOLE correlations to design zero-knowledge proofs. The key idea of VOLEitH is that a GGM-style vector commitment can be adapted to efficiently implement an all-but-one random oblivious transfer functionality, which is then used to construct the core VOLE protocol [16]. Appendix A shows how to realize an all-but-one random oblivious transfer functionality from a GGM tree.

The idea works as follows:

1. The prover locally runs the SoftSpokenOT protocol [16], instantiating the all-but-one random OT with the vector commitment. In particular, the prover generates the GGM trees and sends the commitments to the verifier.

2. Using the correlations derived from the previous step, the parties engage in a zero-knowledge protocol. In particular the prover can embed the secret witness in the random correlations and exploit the homomorphic properties to prove some linear relations.
3. At the end, the verifier simply sends Δ to the prover, who replies with the corresponding openings. The verifier then checks that:
   - the vector commitment openings are correct;
   - the checks inside SoftSpokenOT and the consistency checks are correct;
   - the zero-knowledge verification of the previous point passes.

Intuitively, since the inner ZK protocol is public-coin, the parties can still simulate the protocol without having Δ, and since the proof information is already sent in Step 2, revealing Δ in step 3 does not grant the prover any advantage.

3.3.1 Improving VOLE-in-the-Head for subspace VOLE

The bottleneck of VOLE-in-the-Head is the communication cost due to the VOLE generation procedure. As stated in Section 3.2, to create VOLE correlations over a big field $\mathbb{F}_{p^r}$ we need to compute $\tau = \left\lceil \frac{\lambda}{r \log p} \right\rceil$ repetitions of the basic protocol from SoftSpokenOT. Hence, to use VOLE-in-the-Head in this context, we would need to compute τ tree-vector commitments to generate the small VOLE correlations. However, this results in a major communication overhead, since the commitments output by the tree-vector commitment are very large. To solve this issue, Baum et al. introduce in [18] a batch all-but-one-vector commitment scheme (BAVC) with aborts in the opening phase. Instead of computing τ small trees with $N_1, ..., N_\tau$ leaves, a BAVC allows to compute a single big tree with $\sum_{i=1}^{\tau} N_i$ leaves and open all but τ of these leaves. This approach is more efficient because, with high probability, the paths leading to the hidden leaves will merge relatively close to the leaves, thus reducing the number of internal nodes to be revealed. Even though this number is reduced, it depends on the indexes I of the unopened leaves. Since this dependency would cause some fluctuation in the size of the final signature, a threshold T_{open} is fixed limiting the number of internal nodes to be revealed in the opening phase. If T_{open} is exceeded, then the computation is aborted. In practice, the value of T_{open} is chosen with respect to a trade-off between the size of the commitment and the probability of success of the opening phase.

Remark 2 In the final signature scheme, whenever T_{open} is exceeded in the opening phase, the signer needs to freshly generate the set of indexes I, which is generated as the hash of a challenge `chl`. Hence, the signer uses a counter `ctr` that is incremented every time T_{open} is exceeded, and the new I is computed as the digest of `chl` || `ctr`. Then, `ctr` is included in the signature to let the verifier reconstruct the correct set I.

With this improvement, the cost associated to the tree-vector commitment schemes is reduced from $\sum_{i=1}^{\tau}(\lambda \log_2(N_i) + 2\lambda)$ to $\lambda \cdot T_{\mathsf{open}} + \tau 2\lambda$. A more detailed description

of the BAVC scheme from [18] is found in Appendix A, whereas the procedure for generating VOLE correlations using a BAVC is described in Appendix B.

Remark 3 (Parameters for BAVC) In this work, we will use VOLE correlations defined over $\mathbb{F}_p$ with $p = 127$. Hence, to achieve enough security, we need to employ the subspace VOLE protocol from Section 3.2 with $\tau = \left\lceil \frac{\lambda}{\log_2(p)} \right\rceil$, where $\lambda \in \{128, 192, 256\}$. Therefore, we will use the BAVC scheme to generate a GGM tree with $\tau \cdot 2^{\lceil \log_2(p) \rceil}$ leaves. Our choices for the values of T_{open} are displayed in Table 1.

λ	p	τ	T_{open}
128	127	19	123
192		28	180
256		37	235

Table 1: Values of T_{open} for the different levels of security. The parameters were selected experimentally to obtain a success probability of $\sim 90\%$ (tests run on 5000 instances).

3.4 Zero-knowledge protocols for degree-2 relations

In [3, Section 6.2], the authors propose a public-coin, interactive ZK protocol for degree-2 relations. The idea is to use ℓ VOLE correlations to mask the witness $\mathbf{w}$ and then use Quicksilver [19] to prove that the masked value satisfies the relations. The interactive version of the protocol is described in Algorithm 3.

4 Modellings for PoK for Linear Equivalence

In this section we propose three modellings for proofs of knowledge of a solution of an instance of the linear equivalence problem. We adopt two different approaches. The first two schemes are based on two algebraic modellings for the code equivalence problems. The goal is to show that solving the code equivalence problem is equivalent to solving some quadratic system of equations: in this way we can use $\Pi^t_{2D-\mathsf{Rep}}$ (Algorithm 3) to build a zero-knowledge proof, and thus a VOLEitH signature. Instead, the third scheme takes inspiration from the work of [4], which introduces a VOLEitH signature based on the Permuted Kernel Problem (PKP). In particular, we adapt their idea to handle instances of the linear equivalence problem.

We start with the following proposition:

Proposition 2 *Let $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathbb{F}_p^n$ be two linear codes of dimension k. For each $i \in \{1, 2\}$, let $\mathbf{G}_i$ and $\mathbf{H}_i$ be the generator matrix and the parity-check matrix of code $\mathcal{C}_i$, respectively.*

Algorithm 3 Protocol $\Pi^t_{2D-\mathsf{Rep}}$ [3, Figure 6]

Input:

- both $\mathcal{P}$ and $\mathcal{V}$ hold a set of polynomials $\{g_1, ..., g_t\} \subset \mathbb{F}_p[X_1, ..., X_\ell]_{\leq 2}$;
- $\mathcal{P}$ additionally holds a witness $\mathbf{w} \in \mathbb{F}_p^\ell$ such that $g_i(\mathbf{w}) = 0\ \forall i \in [t]$ and VOLE correlation $\mathbf{u} \in \mathbb{F}_p^{\ell+\tau}, \mathbf{V} \in \mathbb{F}_p^{(\ell+\tau)\times\tau}$

Round 1:

1: $\mathcal{P}$ commits to his witness by sending $\mathbf{d} = \mathbf{w} - \mathbf{u}_{[1,\ell]}$ to $\mathcal{V}$
2: For $i \in [\ell+1, ..., \ell+\tau]$ embed u_i into $\mathbb{F}_{p^\tau}$.
3: For $i \in [\ell]$ embed w_i into $\mathbb{F}_{p^\tau}$

Round 2: $\mathcal{V}$ samples $\chi_i \xleftarrow{\$} \mathbb{F}_{p^\tau}, i \in [t]$ and sends it to $\mathcal{P}$.
Round 3: $\mathcal{P}$ performs the following steps:

4: For each $i \in [t]$ compute

$$\begin{aligned} c_i(Y) &:= \sum_{h\in[0,2]} \bar{g}_{i,h}(v_1 + w_1 \cdot Y, ..., v_\ell + w_\ell \cdot Y) \cdot Y^{2-h} \\ &= \bar{g}_i(w_1, ..., w_\ell) \cdot Y^2 + A_{i,1} \cdot Y + A_{i,0} \end{aligned}$$

where $g_i = g_{i,0} + g_{i,1} + g_{i,2}$, with $\deg(g_{i,h}) = h$ for $h \in [0,2]$, $\bar{g}_i$ denotes the embedding of g_i into $\mathbb{F}_{p^\tau}$ and $A_{i,h}$ are the the aggregated coefficient of Y^h for $h \in \{0,1\}$.

5: Compute

$$u^* = \sum_{i\in[\tau]} u_i X^{i-1} \qquad v^* = \sum_{i\in[\tau]} v_i X^{i-1}$$

where $\mathbb{F}_{p^\tau} \simeq \mathbb{F}_p[X]/F(X)$.

6: Computes $\tilde{a}_0 = \sum_{i\in[t]} \chi_i \cdot A_{i,0} + v^*$ and $\tilde{a}_1 = \sum_{i\in[t]} \chi_i \cdot A_{i,1} + u^*$ and sends them to $\mathcal{V}$

Round 4: $\mathcal{V}$ samples $\mathbf{\Delta} \xleftarrow{\$} \mathbb{F}_{p^r}^\tau$ and sends it to $\mathcal{P}$.
Round 5: $\mathcal{P}$ sends $\mathbf{R} = \mathbf{V} + \mathbf{u}^T[1|...|1]\,\mathrm{diag}(\Delta)$.
Verification: $\mathcal{V}$ does the following:

7: Compute $\mathbf{R}' = \mathbf{R}_{[1,\ell]} + \mathbf{d}^T[1|...|1]\,\mathrm{diag}(\Delta) = \mathbf{V}_{[1,...,\ell]} + \mathbf{w}^T[1, ..., 1]\,\mathrm{diag}(\Delta)$
8: Lift $\mathbf{\Delta}, \mathbf{r}'_1, ..., \mathbf{r}'_\ell, \mathbf{r}_{\ell+1}, ..., \mathbf{r}_{\ell+\tau}$ into $\Delta, r'_1, ..., r'_\ell, r_{\ell+1}, .., r_{\ell+\tau} \in \mathbb{F}_{p^\tau}$.
9: For each $i \in [t]$ compute

$$c_i(\Delta) = \sum_{h\in[0,2]} \bar{g}_{i,h}(r'_1, ..., r'_\ell) \cdot \Delta^{2-h}$$

10: Compute $r^* = \sum_{i\in[\tau]} r_{i+\ell} \cdot X^{i-1}$ such that $r^* = v^* + u^*\Delta$
11: Compute $\tilde{c} = \sum_{i\in[\tau]} \chi_i \cdot c_i(\Delta) + r^*$.
12: Check that $\tilde{c} = \tilde{a}_1\Delta + \tilde{a}_0$.

Finally, let $\mathbf{S} \in \mathsf{GL}_k(p)$ *and* $\mathbf{Q} \in \mathsf{M}_n(p)$*. Then* $\mathbf{G}_2 = \mathbf{S}\mathbf{G}_1\mathbf{Q}$ *if and only if*

$$\mathbf{G}_1\mathbf{Q}\mathbf{H}_2^\top = \mathbf{0}_{k,n-k}. \tag{11}$$

In other words, $\mathcal{C}_1 \overset{LE}{\sim} \mathcal{C}_2$ *if and only if there exists* $\mathbf{Q} \in \mathsf{M}_n(p)$ *satisfying Equation* (11)*.*

Proof Since $\mathcal{C}_1 \overset{\text{LE}}{\sim} \mathcal{C}_2$, we have that there exist $\mathbf{S} \in \mathsf{GL}_k(p)$ and $\mathbf{Q} \in \mathsf{M}_n(p)$ such that

$$\mathbf{G}_2 = \mathbf{S}\mathbf{G}_1\mathbf{Q}.$$

We know that $\mathbf{G}_2\mathbf{H}_2^\top = \mathbf{0}_{k,n-k}$, therefore we have that

$$\mathbf{S}\mathbf{G}_1\mathbf{Q}\mathbf{H}_2^\top = \mathbf{0}_{k,n-k} \implies \mathbf{G}_1\mathbf{Q}\mathbf{H}_2^\top = \mathbf{0}_{k,n-k}$$

Conversely, by assumption $\mathbf{G}_1$,$\mathbf{H}_2$ and $\mathbf{Q}$ are full rank. Hence, $\mathbf{G}_1\mathbf{Q}$ is a generator matrix for the code having $\mathbf{H}_2$ as parity-check matrix. Therefore, there exists an invertible matrix (a change of basis) $\mathbf{S} \in \mathsf{GL}_k(p)$ such that $\mathbf{G}_2 = \mathbf{S}\mathbf{G}_1\mathbf{Q}$. □

Proposition 2 provides a useful relation for the construction of VOLE-based proofs of knowledge for LEP. Its primary advantage is that it does not involve the invertible matrix $\mathbf{S}$. Firstly, the size of the witness greatly reduces, and therefore the communication cost decreases since the prover does not need to directly commit to $\mathbf{S}$ inside the proof. Secondly, since $\mathbf{S}$ has not a precise structure, aside from being invertible, it is more difficult to model inside a zero-knowledge proof.

4.1 Model 1

Notice that Equation (11) provide us a system of equations that $\mathbf{Q}$ should satisfy. However, by itself, this system is not enough as the linear equations are not sufficient to restrict $\mathbf{Q}$ to be a monomial. Indeed, it is possible that a non monomial matrix is a solution for said system, thus we need to add equations to prevent this. In the following we show a possible set of equations that solve the issue.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two linearly equivalent $[n, k]$-linear codes over $\mathbb{F}_p$. Let $\mathbf{G}_1$ denote a generator matrix of $\mathcal{C}_1$, and let $\mathbf{H}_2$ be a parity-check matrix of $\mathcal{C}_2$. Our goal is to provide a system of quadratic equations that, together with the one from Equation (11), has as only solution the monomial matrix $\mathbf{Q}$ such that $\mathbf{G}_1\mathbf{Q}\mathbf{H}_2^\top = \mathbf{0}_{k,n-k}$.

The unknown entries of $\mathbf{Q}$ are denoted by the n^2 variables $\{x_{i,j}\}_{i,j\in[n]}$. As said, the first set of equation $\mathcal{S}_1$ is trivially given by the relation $\mathbf{G}_1\mathbf{Q}\mathbf{H}_2^\top = \mathbf{0}_{k,n-k}$ of Equation (11):

$$\mathcal{S}_1 := \left\{ \sum_{\ell\in[n]} \left(\sum_{m\in[n]} (\mathbf{G}_1)_{i,m} \cdot x_{m,\ell} \right) \cdot (\mathbf{H}_2)_{j,\ell} = 0 \mid i \in [k], j \in [n-k] \right\} \tag{12}$$

The second and third sets of equations are used to ensure that $\mathbf{Q}$ is a monomial matrix. To define this set, we start by observing that the matrix $\mathbf{Q} = [x_{i,j}]_{i,j\in[n]}$ can be described by the following set of equations:

$$x_{i_1,j} \cdot x_{i_2,j} = 0 \text{ for } i_1, i_2, j \in [n], i_1 \neq i_2 \tag{13}$$

$$x_{i,j_1} \cdot x_{i,j_2} = 0 \text{ for } i, j_1, j_2 \in [n], j_1 \neq j_2 \tag{14}$$

Indeed, Equation (13) ensures that each column of $\mathbf{Q}$ has at most 1 entry different from 0, i.e. that the Hamming weight of each column is at most 1. Similarly Equation (14) ensures that each row has Hamming weight at most 1. However, this set of equations admits as solution any matrix having zero rows or columns. To prevent this, we consider $2n$ additional unknowns $\{w_j\}_{j\in[n]}$ and $\{\widetilde{w}_i\}_{i\in[n]}$, and the following quadratic equations:

$$w_j \cdot \sum_{i\in[n]} x_{i,j} = 1 \text{ for } j \in [n] \tag{15}$$

$$\widetilde{w}_i \cdot \sum_{j\in[n]} x_{i,j} = 1 \text{ for } i \in [n] \tag{16}$$

These equations impose that the sum of the elements on each column and on each row of $\mathbf{Q}$ is invertible, and thus it is different from 0. In other words, Equations (15) and (16) ensure that the columns and the rows of $\mathbf{Q}$ have Hamming weight at least 1.

It could be tempting to avoid the additional variables w_j by simply showing $\sum_{i\in[n]} x_{i,j} \neq 0$ (and the same for the columns). However, doing so is either not secure or computationally heavier than this solution: indeed we either need to show the exact non zero value or to use higher degree relations (e.g. $\sum_{i\in[n]} x_{i,j}^p = 1$) that requires additional VOLE correlations.

Proposition 3 *$\mathbf{Q} = [x_{i,j}]$ is a monomial matrix if and only if there exist $\{w_j\}_{j\in[n]}, \{\widetilde{w}_i\}_{i\in[n]}$ such that the equations 13, 14, 15 and 16 are satisfied.*

Proof By construction any monomial matrix satisfies these equations, and the values $\{w_j\}_{j\in[n]}, \{\widetilde{w}_i\}_{i\in[n]}$ are equal to the inverses of the non-zero constants on each row and column of that matrix.

To prove the other direction, notice that if $\mathbf{Q}$ satisfies Equations (13) and (14), then its rows and columns have Hamming weight at most 1. Further, if $\mathbf{Q}$ also satisfies Equations (15) and (16), then its rows and columns have Hamming weight at least 1, since at least one element on each row and column must be invertible (and thus non zero). Putting these results together, we obtain that the rows and columns of $\mathbf{Q}$ have Hamming weight equal to 1, and thus that $\mathbf{Q}$ is a monomial matrix. □

It is immediate to see that the system of equations by Equations (13) to (16), is redundant, and can be minimized as shown in the following result.

Proposition 4 *Let $\mathbf{Q} = [x_{i,j}]_{i,j\in[n]}$ be an $n \times n$ matrix with coefficients in $\mathbb{F}_p$. Then $\mathbf{Q}$ is a monomial matrix if and only if there exist $\{w_j\}_{j\in[n]} \subset \mathbb{F}_p$ such that the following sets of equations are satisfied*

$$\mathcal{S}_2 := \left\{ x_{i,j_1} \cdot x_{i,j_2} = 0 \mid i, j_1, j_2 \in [n], j_1 \neq j_2 \right\} \tag{17}$$

$$\mathcal{S}_3 := \left\{ w_j \cdot \sum_{i \in [n]} x_{i,j} = 1 \mid j \in [n] \right\} \tag{18}$$

Proof One direction of the proof follows directly from Proposition 3. Indeed, if $\mathbf{Q}$ is a monomial matrix, then it satisfies both $\mathcal{S}_2$ and $\mathcal{S}_3$ since they are a subset of the equations of Proposition 3 .

For the other direction, on one hand $\mathcal{S}_2$ implies that each row of $\mathbf{Q}$ has at most one element different from 0, and thus that $\mathbf{Q}$ has at most n non-zero elements. On the other hand, $\mathcal{S}_3$ imposes that each column of $\mathbf{Q}$ is non-zero, and thus that $\mathbf{Q}$ has at least n non-zero elements. Together, these two statements imply that $\mathbf{Q}$ has exactly n entries different from 0. Furthermore, these elements are placed such that each row and each column have exactly one entry different from 0. Indeed:

- if $\mathbf{Q}$ has a zero row, then there exists a row having more than one element different from 0, contradicting the equations in $\mathcal{S}_2$;
- if $\mathbf{Q}$ has a column with more than one non-zero entry, then $\mathbf{Q}$ also has a zero column, contradicting the constraints imposed by $\mathcal{S}_3$.

Therefore, $\mathbf{Q}$ is a monomial matrix.

□

Remark 4 An equivalent result is obtained by considering the sets of equations:

$$\widetilde{\mathcal{S}}_2 := \left\{ x_{i_1,j} \cdot x_{i_2,j} = 0 \mid i_1, i_2, j \in [n], i_1 \neq i_2 \right\}$$

$$\widetilde{\mathcal{S}}_3 := \left\{ \widetilde{w}_i \cdot \sum_{j \in [n]} x_{i,j} = 1 \mid i \in [n] \right\}$$

Since $\mathbf{Q}$ is a square matrix there is no difference in the number of equations or variables needed, so the two sets are perfectly equivalent also from a performance standpoint.

Remark 5 The set $\mathcal{S}_2$ is composed by $n \cdot \frac{n(n-1)}{2}$ equations in the n^2 unknowns $\{x_{i,j}\}_{i,j \in [n]}$. The set $\mathcal{S}_3$ is composed by n equations in the n^2 unknowns $\{x_{i,j}\}_{i,j \in [n]}$, plus the n additional variables $\{w_m\}_{m \in [n]}$.

Thus, $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ form a system of $k \cdot (n-k) + n \cdot \frac{n(n-1)}{2} + n$ equations in $n^2 + n$ unknowns.

In summary, the sets of equations $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ provide a quadratic system of equations imposing that its solution is a monomial matrix $\mathbf{Q} \in \mathsf{M}_n(p)$ satisfying $\mathbf{G}_1 \mathbf{Q} \mathbf{H}_2 = \mathbf{0}_{k,n-k}$. Thus, we can prove the following reduction:

Proposition 5 *The map* $(\mathbf{G}_1, \mathbf{H}_2) \mapsto \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$ *is a polynomial-time reduction from LEP to MQ. In particular, if* $\{\{\overline{x}_{i,j}\}_{i,j \in [n]}, \{\overline{w}_m\}_{m \in [n]}\}$ *is a solution of the MQ instance* $\{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$*, then* $\overline{\mathbf{Q}} = [\overline{x}_{i,j}]_{i,j \in [n]}$ *is a solution to the LEP instance* $(\mathbf{G}_1, \mathbf{H}_2)$*.*

Proof The proof is an immediate consequence of Proposition 2 and Proposition 4. Indeed, if $\mathbf{Q}$ is a solution for the LEP instance $(\mathbf{G}_1, \mathbf{H}_2)$ then it should satisfy the equations in $\mathcal{S}_1$ by definition and, since $\mathbf{Q}$ is a monomial matrix, it should also satisfy the equations in $\mathcal{S}_2, \mathcal{S}_3$. The vice-versa follows trivially. □

As a direct consequence of Proposition 5 we have that showing the knowledge of $\mathbf{Q}$ is equivalent to showing the knowledge of a solution of the system of equations of $\mathcal{S}_1, \mathcal{S}_2$ and $\mathcal{S}_3$ (that can therefore be made public). This allows us to directly apply the construction of Algorithm 3 to obtain a VOLE protocol for the knowledge of $\mathbf{Q}$.

Remark 6 We mention that in [10, Section 4.2], the author proposes a similar model while studying algebraic approaches to solve the Linear Code Equivalence Problem (which in [10] is referred to as Diagonal Code Equivalence Problem). However, the author employs a different system of equations to ensure that all the columns of $\mathbf{Q}$ are non-zero (i.e. in place of our set $\mathcal{S}_3$). In particular, the author uses the following set of equations

$$\prod_{k\in[p-1]} \left(\sum_{i\in[n]} x_{i,j} - \alpha^k \right) = 0 \text{ for } j \in [n]$$

where $\alpha \in \mathbb{F}_p$ is a primitive element of $\mathbb{F}_p$, i.e $\mathbb{F}_p^* = \langle \alpha \rangle$.

However, these equations have a degree higher than 2, thus they are not suitable for our purpose.

4.2 Model 2

The main drawback of Model 1 is that it needs $O(n^3)$ equations and thus fails short in improving LESS computational complexity. In particular, this is due to the set $\mathcal{S}_2$, which is the only one having $O(n^3)$ elements. We now show a second algebraic modelling, that builds on the same initial idea of Model 1 but reduces the number of equations needed. As before, given an instance of LEP, we want to prove the knowledge of a monomial matrix $\mathbf{Q}$ that satisfies the identity of Proposition 2. To do this, we keep the sets of equations $\mathcal{S}_1$ and $\mathcal{S}_3$, described in Equations (12) and (18) in the previous section, unchanged, and we introduce a new set of equations $\mathcal{S}_4$ to impose that the rows of $\mathbf{Q}$ have Hamming weight at most 1. The new set of equations $\mathcal{S}_4$ is built by adapting the quadratic modelling for the syndrome decoding problem introduced in [20, Modelling 5] to prove that each row of $\mathbf{Q}$ has Hamming weight at most 1.

Let $\ell = \lfloor \log_p(n) \rfloor + 1$, for each $i \in [n]$, let $\mathbf{x}_i = (x_{i,1}, ..., x_{i,n})$ denote the vector of variables representing the i-th row of $\mathbf{Q}$. Also, for each $i \in [n]$ we introduce the sets of variables $\mathbf{Y}^i = (\mathbf{Y}^i_1, ..., \mathbf{Y}^i_n)$, where $\mathbf{Y}^i_j = (y^i_{j,1}, ..., y^i_{j,\ell})^\top$ is a column vector of ℓ variables, and $\mathbf{Z}_i = (z_{i,1}, ..., z_{i,n})$, which only take values in $\{0, 1\}$. Finally, define the vector $\mathbf{y}_0 = (1, 0, ..., 0)^\top$. Following the approach of [20], for each $i \in [n]$ we introduce the following sets of equations:

1. the $2n$ quadratic *support constraint equations*:

$$T^i_1 := \{x_{i,j}(z_{i,j} - 1) = 0 \mid j \in [n]\} \cup \{z^2_{i,j} - z_{i,j} = 0 \mid j \in [n]\} \tag{19}$$

This set of equations identifies the non-zero entries of the vectors $\mathbf{x}_i$. The right set in the union ensures us that all the $z_{i,j}$ are in $\{0,1\}$, while the left one provides the links between $x_{i,j}$ and $z_{i,j}$: indeed, for each $x_{i,j}$ we have that $z_{i,j} = 1$ if $x_{i,j} \neq 0$. In particular, it follows that $\mathsf{wt}(\mathbf{Z}_i) \geq \mathsf{wt}(\mathbf{x}_i)$ for all $i \in [n]$.

2. the $n \cdot \ell$ equations from the *Hamming weight computation encoding*:

$$T_2^i := \{\mathbf{Y}_1^i - (1 - z_{i,1}) \cdot \mathbf{y}_0 - z_{i,1} \cdot \mathbf{C}_f \cdot \mathbf{y}_0 = 0\} \cup \\ \cup \{\mathbf{Y}_j^i - (1 - z_{i,j}) \cdot \mathbf{Y}_{j-1}^i - z_{i,j} \cdot \mathbf{C}_f \cdot \mathbf{Y}_{j-1}^i = 0 \mid j \in [2, n]\} \quad (20)$$

where $\mathbf{C}_f$ is the companion matrix of a primitive polynomial of degree ℓ over $\mathbb{F}_p$. This set of equations is used to model the weight of each row: in particular, for each $1 \leq t \leq n$, the variable $\mathbf{Y}_j^i$ represent the weight of the vector $(z_{i,1}, ..., z_{i,j})$. In a nutshell, the variables $\mathbf{Y}_j^i$ correspond to the possible states of a linear recurrence of order n defined by starting from the fixed value $\mathbf{y}_0$, i.e. $\{\mathbf{y}_0, \mathbf{C}_f\mathbf{y}_0, \mathbf{C}_f^2\mathbf{y}_0, \ldots\}$. By Equation (20), whenever $z_{i,j}$ is non-zero, the corresponding variable $\mathbf{Y}_j^i$ is different from $\mathbf{Y}_{j-1}^i$, whereas, if $z_{i,j} = 0$ we have $\mathbf{Y}_j^i = \mathbf{Y}_{j-1}^i$ (namely, the value of $\mathbf{Y}_{j-1}^i$ is not updated). Therefore, if the set $\{z_{i,j}\}_{j=1,\ldots,n}$ has exactly t non-zero elements, the variable $\mathbf{Y}_n^i$ will be *updated* exactly t times, hence we would have $\mathbf{Y}_n^i = \mathbf{C}_f^t \cdot \mathbf{y}_0$. Thus, to impose that exactly t among the n elements in $\{z_{i,j}\}_{j=1,\ldots,n}$ are non-zero (i.e., to impose that the Hamming weight of $\{x_{i,j}\}_{j=1,\ldots,n}$ is bounded by t) we could simply add a last equation of the form $\mathbf{Y}_n^i = \mathbf{C}_f^t\mathbf{y}_0$. This last equation is not present in Equation (20), but it is instead described in the next set of equations, specialized to the case of $t = 1$, i.e. to the case in which, for any i, the i-th row of the matrix has weight 1. We refer the interested reader to [20, Proposition 2] for the proof of the correctness of this modeling.

3. the ℓ equations from the *weight constraint encoding* where we bind the weight of each row to be equal to 1:

$$T_3^i := \{\mathbf{Y}_n^i - \mathbf{C}_f\mathbf{y}_0 = 0\} \quad (21)$$

This set of equations forces the vector $(z_{i,1}, ..., z_{i,n})$ to have weight exactly 1 (notice that, from the previous point, the variables $\mathbf{Y}_i^n$ represent the weight of the vector $(z_{i,1}, ..., z_{i,n})$). Thanks to the set T_1^i, it also implies that the vector $(x_{i,1}, ..., x_{i,n})$ has weight 1.

Therefore, for each $i \in [n]$ we obtain the set of equations

$$\mathcal{T}_i := T_1^i \cup T_2^i \cup T_3^i$$

composed by $2n + \ell(n + 1)$ equations in $n(\ell + 2)$ unknowns. Finally, the set $\mathcal{S}_4$ is defined as

$$\mathcal{S}_4 := \bigcup_{i=1}^{n} \mathcal{T}_i \quad (22)$$

and is composed by $n \cdot (2n + \ell(n + 1))$ equations in $n^2(\ell + 2)$ unknowns.

Proposition 6 *Let $\mathbf{Q} = [x_{i,j}]_{i,j\in[n]}$ be an $n \times n$ matrix with coefficients in $\mathbb{F}_p$. Then, $\mathbf{Q}$ is a monomial matrix if and only if there exist $\{w_j\}_{j\in[n]} \subset \mathbb{F}_p$, $\{\mathbf{z}_1, ..., \mathbf{z}_n\} \subset \{0,1\}^n$ and $\{\mathbf{y}_1^i, ..., \mathbf{y}_n^i \mid i \in [n]\} \subseteq \mathbb{F}_p^\ell$ such that the equations in $\mathcal{S}_3$ (18) and $\mathcal{S}_4$ (22) are satisfied.*

Proof If $\mathbf{Q}$ is a monomial matrix, by setting $\{w_j\}_{j\in[n]}$ to be the inverses of the non-zero elements on each column of $\mathbf{Q}$ we obtain that the equations in $\mathcal{S}_3$ are satisfied. Moreover, for each row $\mathbf{x}_i$ of $\mathbf{Q}$ we can define $\mathbf{z}_i = (z_{i,1}, ..., z_{i,n})$ as

$$\begin{cases} z_{i,j} = 1 \text{ if } x_{i,j} \neq 0 \\ z_{i,j} = 0 \text{ if } x_{i,j} = 0 \end{cases}$$

Then, for each $i \in [n]$, the pair $(\mathbf{x}_i, \mathbf{z}_i)$ trivially satisfies the equations in T_1^i. Also, for each $i \in [n]$, $\mathsf{wt}(\mathbf{z}_i) = \mathsf{wt}(\mathbf{x}_i) = 1$. Hence, $\mathbf{z}_i$ uniquely determines a solution $(\mathbf{y}_1^i, ..., \mathbf{y}_n^i, \mathbf{z}_i)$ of the equations in T_2^i and T_3^i. In other words, for each row $\mathbf{x}_i$ of $\mathbf{Q}$ we are able to determine a solution of the system of equations defined by $\mathcal{T}_i$; thus, we obtain that the equations in $\mathcal{S}_4$ are satisfied.

For the other direction, we already know that the equations in $\mathcal{S}_3$ impose that the columns of $\mathbf{Q}$ have at least one non-zero entry. To prove that $\mathbf{Q}$ is a monomial matrix, it suffices to prove that the equations in $\mathcal{S}_4$ impose that each row of $\mathbf{Q}$ has at most one non-zero entry. Since $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,n})$ and $\mathbf{z}_i = (z_{i,1}, \ldots, z_{i,n})$ satisfy the equations in Equation (19) for all $i \in [n]$, we know that if $x_{i,j} \neq 0$ then $z_{i,j} = 1$ for all $i, j \in [n]$. In particular, this implies that $\mathsf{wt}(\mathbf{z}_i) \geq \mathsf{wt}(\mathbf{x}_i)\ \forall i \in [n]$. Then, as discussed above, for all $i \in [n]$ the equations in T_2^i (Equation (20)) impose that $\mathbf{Y}_n^i = \mathbf{C}_f^t \mathbf{y}_0$, with $t = \mathsf{wt}(\mathbf{z}_i)$. Since the values $\{\mathbf{y}_1^i, \ldots, \mathbf{y}_n^i\}$,$\mathbf{z}_i$ satisfy T_2^i, from the discussion above it follows that

$$\mathbf{y}_n^i = \mathbf{C}_f^t \mathbf{y}_0, \tag{23}$$

with $t = \mathsf{wt}(\mathbf{z}_i)$. Lastly, Equation (23) together with T_3^i (Equation (21)) implies that

$$t = \mathsf{wt}(\mathbf{z}_i) = 1, \tag{24}$$

and thus $\forall i \in [n]\ \mathsf{wt}(\mathbf{x}_i) \leq \mathsf{wt}(\mathbf{z}_i) = 1$, meaning that each $\mathbf{x}_i$ (and hence each row of $\mathbf{Q}$)) has at most 1 non-zero entry. □

Remark 7 To ensure that the equations in T_2^i (Equation (20)) correctly encode the weight of the vector $\mathbf{z}_i$, we must choose an irreducible polynomial f such that $f(0) \neq 0$ and $\mathsf{ord}(f) = \mathsf{ord}(\mathbf{C}_f) > \mathsf{wt}(z)$. Indeed, from the theory of linear recurring sequences, it follows that, for a non zero initial vector $\mathbf{y}_0$, the sequence $\mathbf{y}_0, \mathbf{C}_f\mathbf{y}_0, \mathbf{C}_f^2\mathbf{y}_0, \ldots$ is periodic of least period $\mathsf{ord}(f)$. In other words, this ensures that $\mathbf{C}_f^i\mathbf{y}_0 \neq \mathbf{C}_f^j\mathbf{y}_0$ for all $i, j < \mathsf{ord}(f),\ i \neq j$.

In summary, the sets of equations $\mathcal{S}_1$ (12), $\mathcal{S}_3$ (17) and $\mathcal{S}_4$ (22) provide a quadratic system of equations whose solution determines the existence of a monomial matrix $\mathbf{Q} \in \mathsf{M}_n(p)$ such that $\mathbf{G}_1\mathbf{Q}\mathbf{H}_2^\top = \mathbf{0}_{k,n-k}$. The second reduction from LEP to MQ is thus formulated as follows.

Proposition 7 *The map $(\mathbf{G}_1, \mathbf{H}_2) \mapsto \{\mathcal{S}_1, \mathcal{S}_3, \mathcal{S}_4\}$ is a polynomial-time reduction from LEP to MQ. In particular, if*

$$\left\{\{\overline{x}_{i,j}\}_{i,j\in[n]}, \{\overline{w}_j\}_{j\in[n]}, \{\overline{\mathbf{z}}_i \mid i \in [n]\}, \{\overline{\mathbf{y}}_1^i, ..., \overline{\mathbf{y}}_n^i \mid i \in [n]\}\right\}$$

is a solution of the MQ instance $\{\mathcal{S}_1, \mathcal{S}_3, \mathcal{S}_4\}$*, then* $\overline{\mathbf{Q}} = [\overline{x}_{i,j}]_{i,j\in[n]}$ *is a solution to the LEP instance* $(\mathbf{G}_1, \mathbf{H}_2)$*.*

Proof The proof follows directly from Proposition 2 and Proposition 6. □

Remark 8 Overall, $\mathcal{S}_1, \mathcal{S}_3$ and $\mathcal{S}_4$ form a system of $k(n-k)+n+n\cdot(2n+\ell(n+1))$ equations in $n^2\cdot(\ell+2)+n$, with $\ell = \log_p(n)$. In particular, the set $\mathcal{S}_4$ has only $n^2(\log_p(n)+n)$ equations, compared to the $n\cdot\frac{n(n-1)}{2}$ equations of $\mathcal{S}_2$. This lowers the overall complexity from $O(n^3)$ to $O(n^2\log_p n)$.

4.3 Model 3

In this section, we describe our last model of PoK for the linear equivalence problem. Unlike the previous ones, this model is not based on an MQ reduction, but instead it relies on higher degree relations: then, similarly to what is done for the first two modellings, the knowledge of a solution to the polynomial constraints can be proved using the protocol from [3, Appendix B.1]. In Appendix C we provide an alternative description of the model, that is more in line with the original work of [4].

Notation For shortness, we will write $d = \lceil \log_2(n) \rceil$. We will denote $\mathcal{B}^j \in \{0,1\}^d$ the binary representation of an integer $j \in [0, n-1]$, and by $\overline{\mathcal{B}^j}$ its complement.

We start by reporting the techniques from [4]. In the work of Bettaieb et al., the permutation matrix $\mathbf{P}$ is represented by n integers $\mathsf{pos}_0, ..., \mathsf{pos}_{n-1} \in [0, n-1]$, which correspond to the positions of the non-zero entries in each row of $\mathbf{P}$. Then, random VOLE correlations are used to reconstruct each row of $\mathbf{P}$, by computing one commitment for each element of the rows.

The core idea is the observation that, given two binary $\mathbf{b}^1, \mathbf{b}^2$, $\mathbf{b}^1 \oplus \mathbf{b}^2 = (1, ..., 1)^\top$ if and only if $\mathbf{b}^2 = \overline{\mathbf{b}^1}$. As a consequence, the product of the entries of $\mathbf{b}^1 \oplus \mathbf{b}^2$ is equal to 1 when $\mathbf{b}^2 = \overline{\mathbf{b}^1}$ and 0 otherwise. Given a integer $\mathsf{pos} \in [0, n-1]$, let $\mathbf{p} = \mathcal{B}^{\mathsf{pos}} \in \mathbb{F}_2^d$. Then, the vector $\mathbf{z} \in \mathbb{F}_2^n$ defined as

$$\forall j \in [0, n-1].\ z_j := \prod_{k=0}^{d-1} (\overline{\mathcal{B}^j}_k \oplus p_k) \tag{25}$$

contains exactly one entry equal to 1 at position $j = \mathsf{pos}$, while all its other coordinates are equal to 0. In other words, this procedure reconstructs an elementary vector z given the position of the 1.

Unfortunately, the procedure cannot be directly applied when working over a generic $\mathbb{F}_p$ instead of $\mathbb{F}_2$. Indeed, there are two problems. First, there is no guarantee that the vectors are binary: while this comes for free when working in $\mathbb{F}_2$, additional checks are needed to force the prover to only use binary vectors in $\mathbb{F}_p$. Second, when working

in odd characteristic $-1 \neq 1$, thus to implement the XOR we need a higher degree relation. Indeed, given $\mathsf{pos} \in [0, n-1]$, let $\mathbf{p} = \mathcal{B}^{\mathsf{pos}} \in \mathbb{F}_p^d$ and define its complement $\overline{\mathcal{B}^{\mathsf{pos}}}$ to be the vector such that $\mathbf{p} + \overline{\mathcal{B}^{\mathsf{pos}}} = (1, \ldots, 1)^\top \in \mathbb{F}_p^d$. Then, in this case,

$$\prod_{k=0}^{d-1} (p_k - \overline{\mathcal{B}^{\mathsf{j}}}_k) = \begin{cases} \pm 1 \text{ if } j = \mathsf{pos} \\ 0 \text{ otherwise} \end{cases}$$

Hence, to reconstruct the elementary vector $\mathbf{z} \in \mathbb{F}_p^n$ containing a 1 at position $j = \mathsf{pos}$, its entries must be defined as

$$\forall j \in [0, n-1].\ z_j := \prod_{k=0}^{d-1} (p_k - \overline{\mathcal{B}^{\mathsf{j}}}_k)^2,$$

with the squaring enforcing $z_{\mathsf{pos}} = 1$.

Following this reasoning, the secret monomial matrix $\mathbf{Q} = \mathbf{DP}$ is represented by the following variables:

- $n \cdot d$ variables $\{p_{i,j}\}_{i \in [0,n-1], j \in [0,d-1]}$ representing the row-wise positions of the non-zero entries of $\mathbf{P}$: for each $i \in [0, n-1]$, the d variables $\{p_{i,j}\}_{j \in [0,d-1]}$ represent the binary expansion of the position pos_i of the non-zero entry in the i-th row of $\mathbf{P}$;
- n variables $\{c_i\}_{i \in [0,n-1]}$ representing the values on the diagonal of $\mathbf{D}$;
- n variables $\{\overline{c}_i\}_{i \in [0,n-1]}$ equal to the inverses of the values represented by the variables $\{c_i\}_{i \in [0,n-1]}$.

Then, Model 3 is realized by the following sets of polynomial constraints:

- $n \cdot d$ equations of degree 2 checking that the binary representations of $\mathsf{pos}_0, \ldots, \mathsf{pos}_{n-1}$ are indeed binary vectors over $\mathbb{F}_p$

$$\mathcal{S}_{\mathsf{pos}} = \{\, p_{i,j}(p_{i,j} - 1) = 0 \mid\ i \in [0, n-1], j \in [0, d-1] \} \tag{26}$$

- n equations of degree 2 imposing that the elements on the diagonal of $\mathbf{P}$ are non-zero

$$\mathcal{S}_{\mathsf{const}} = \{ c_i \cdot \overline{c}_i - 1 = 0 \ \mid i \in [0, n-1] \} \tag{27}$$

- n equations of degree $2d$ imposing that each column of $\mathbf{Q}$ has exactly one non-zero entry

$$\mathcal{S}_{\mathsf{col}} = \left\{ \left(\sum_{i=0}^{n-1} Z(i,j) \right) - 1 = 0 \ \mid i \in [0, n-1] \right\} \tag{28}$$

  where $\forall i \in [0, n-1], j \in [0, n-1]$ the parameter $Z(i,j)$ is defined as

$$Z(i,j) := \left(\prod_{m=0}^{d-1} \left(p_{i,m} - \overline{\mathcal{B}^{j}}_m \right) \right)^2. \tag{29}$$

By construction, for all $i \in [0, n-1]$ the variables $(Z(i,0), \ldots, Z(i,n-1))$ encode the i-th row of the permutation matrix $\mathbf{P}$: this follows from the construction of Equation (25). Indeed, for all $i \in [0, n-1]$, $Z(i,j)$ is equal to 1 if $j = \mathsf{pos}_i$, whereas $Z(i,j) = 0$ whenever $j \neq \mathsf{pos}_i$.

– $k \cdot (n-k)$ equations of degree $2d+1$ imposing that the secret $\mathbf{Q} = \mathbf{DP}$ satisfies the equation $\mathbf{G}_1\mathbf{Q}(\mathbf{H}_2)^\top = \mathbf{0}_{k,n-k}$

$$\mathcal{S}_{\mathsf{eq}} = \left\{ \sum_{m=0}^{n-1} \left((\mathbf{G}_1)_{i,m} \cdot c_m \sum_{\ell=0}^{n-1} Z(m,\ell) \cdot (\mathbf{H}_2)_{j,\ell} \right) = 0 \mid i \in [0,k-1], j \in [0,n-k-1] \right\} \tag{30}$$

Remark 9 Overall, the sets $\mathcal{S}_{\mathsf{pos}}, \mathcal{S}_{\mathsf{const}}, \mathcal{S}_{\mathsf{col}}, \mathcal{S}_{\mathsf{eq}}$ form a system of $3n + k(n-k)$ polynomial constraints of degree maximum $2d+1$ in $n \cdot d + 2n$ variables.

Proposition 8 *Let $(\mathbf{G}_1, \mathbf{H}_2)$ be an instance of LEP. A monomial matrix $\mathbf{Q}$ is a solution for it if and only if it satisfy the set of equations $\mathcal{S}_{\mathsf{pos}}, \mathcal{S}_{\mathsf{const}}, \mathcal{S}_{\mathsf{col}}, \mathcal{S}_{\mathsf{eq}}$.*

Proof We first show that a solution for the LEP problem satisfy the system of equations. Let $\mathbf{Q}$ be a monomial matrix. For all $1 \leq i \leq n$ let $\mathbf{p}_i$ the vector containing the binary expansion of the non-zero position of the $i-$th row. Trivially $p_{i,j}$ satisfy $\mathcal{S}_{\mathsf{pos}}$. Moreover, from the observation above, they also satisfy $\mathcal{S}_{\mathsf{col}}$ Now let q_i be the non zero elements of $\mathbf{Q}$, by definitions q_i has an inverse $\bar{q}_i$ for all i and thus $(q_i, \bar{q}_i)_{i \in \{1,\ldots,n\}}$ are a solution for $\mathcal{S}_{\mathsf{const}}$. Finally, the last set of equations ensure that $\mathbf{Q}$ is a solution for $\mathbf{G}_1\mathbf{Q}\mathbf{H}_2^\top = \mathbf{0}$.

We now do the converse direction. Let $\{\mathbf{p}_i\}_{i \in \{1,\ldots,n\}}$ be a solutions for the set $\mathcal{S}_{\mathsf{pos}}$. By construction we have that they are binary vectors and by construction each of them represent an integer in $\{1, ..., n\}$. Since they are also solutions of $\mathcal{S}_{\mathsf{col}}$ we have that each integer in $\{1, ..., n\}$ is represented by one of the $\mathbf{p}_i$. By the pigeonhole principle they must be all different, thus they represent the non-zero position of a permutation matrix $\mathbf{P}^*$. Notice that equation $\mathcal{S}_{\mathsf{const}}$ force all the c_m in $\mathcal{S}_{\mathsf{eq}}$ to be non-zero, thus we have that the matrix $(c_1, ..., c_n)\mathbf{P}^*$ is a monomial matrix that solve the given LEP instance.

□

5 VOLEitH Signature from LEP

In this section, we present the VOLEitH signatures based on LEP obtained from the VOLE-based PoK introduced in Section 4.

Remark 10 As in [2], the verification of the ZK proofs for degree-m relations, with $m = 2$ for Model 1 and 2 and $m = 2d+1$ for Model 3, is slightly modified. For instance, in Algorithm 3 the prover sends the verifier 2 values $\tilde{a}_1, \tilde{a}_0$, which are used to carry out the verification step; in general, as shown in the protocol from [3, Appendix B.1], when proving the knowledge of a solution to a system of degree-m equations, the prover sends the verifier m coefficients $\tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_{m-1}$. In the signature scheme, instead of sending all m coefficients, the signer sends $m-1$ of them, namely $\tilde{a}_1, \ldots, \tilde{a}_{m-1}$: then, the verifier computes the value $\tilde{c}$ as in

Algorithm 3 (and similarly in [3, Appendix B.1]), and uses it to recompute $\tilde{a}_0$, as shown in Algorithm 5 - Line 5.

Briefly, the signature scheme is defined by the following algorithms:

- **Key Generation:** The signer picks a random code $\mathcal{C}$ with generator matrix $\mathbf{G}_1$ and a random monomial matrix $\mathbf{Q}$. The private key is $\mathbf{Q}$, the public key is $(\mathbf{G}_1, \mathbf{H}_2)$, where $\mathbf{H}_2$ is the parity check matrix of the code having $\mathbf{G}_1\mathbf{Q}$ as generator matrix.
- **Signature Algorithm:** The signer transforms the public key in a system of degree-m equations as per Section 4 and exploits the knowledge of $\mathbf{Q}$ to obtain a solution $\mathbf{w}$ for it. Then, the signer computes the VOLE correlations as per Sections 3.1 and 3.2. The signature algorithm is obtained by applying the Fiat-Shamir transform to the chosen PoK (among those described in Section 4).
- **Verification Algorithm:** To verify the signature, the receiver recomputes $\widetilde{V}$, a_0 and Δ using the information stored in the signature σ, and then checks that the VOLE consistency check and the verification of the PoK hold.

The complete algorithm is shown in Algorithm 4 and Algorithm 5.

Algorithm 4 Protocol $\mathsf{Sign}_{\mathsf{VOLE-LEP}}(\mathtt{m}, \mathtt{sk}, \mathtt{pk})$

Input:

- the message $\mathtt{m}$ to be signed
- the public key $\mathtt{pk} = (\mathbf{G}_1, \mathbf{H}_2)$;
- the private key the monomial matrix $\mathtt{sk} = \mathbf{Q}$ such that $\mathbf{G}_1\mathbf{Q}\mathbf{H}_2 = \mathbf{0}$.

Sign:

1: Compute the system of degree-m equations $\{g_1, ..., g_t\} \in \mathbb{F}_p[X_1, ..., X_\ell]$ from $\mathtt{pk}$ and a solution $\mathbf{w} \in \mathbb{F}_p^\ell$ for it from $\mathtt{sk}$ as per Section 4.

2: Sample $r \xleftarrow{\$} \{0,1\}^\lambda$, compute $\mathtt{iv} \leftarrow \mathsf{H}(\mathtt{sk}||\mathtt{pk}||\mathtt{m}||r)$ and generate VOLE correlations $(\mathtt{com}, \mathtt{decom}, \mathbf{c}_1, ..., \mathbf{c}_{\tau-1}, \mathbf{u}, \mathbf{V}) := \mathsf{VOLECommit}(r, \mathtt{iv}, \ell + m\tau)$.

3: Set $\mathtt{chl}_1 \leftarrow \mathsf{H}(\mathtt{pk}||\mathtt{com}||\mathbf{c}_1||...||\mathbf{c}_{\tau-1})$.

4: Run the VOLE consistency check of [17] to check that all the VOLE correlations are related to the same $\mathbf{u}$ with challenge $\mathtt{chl}_1$, obtaining $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{V}}$.

5: Embed the witness $\mathbf{w}$ into $\mathbf{u}$ as $\mathbf{d} = \mathbf{w} \oplus \mathbf{u}_{[0,\ell-1]}$

6: Run the protocol from Model i on input $\mathtt{sk}, \mathtt{pk}$ and VOLE correlations $\mathbf{u}$ and $\mathbf{V}$ to obtain $\tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_{m-1}$. The challenges $(\chi_i)_{i\in[t]}$ are computed from $\mathtt{chl}_2 = \mathsf{H}(\mathtt{chl}_1||\tilde{\mathbf{u}}||\tilde{\mathbf{V}}||\mathbf{d})$.

7: Set $\mathtt{ctr} = 0$, $\mathtt{chl}_3 := \mathsf{H}(\mathtt{chl}_2||\tilde{a}_0||\tilde{a}_1||\cdots||\tilde{a}_{m-1}||\mathtt{ctr})$ and compute $I = \mathsf{GetChallenge}(\mathtt{chl}_3)$: if $I = \perp$, then increase $\mathtt{ctr}$ and repeat step 7.

8: Compute $\mathtt{decom}_I := \mathsf{BAVC.Open}(\mathtt{decom}, I, \mathtt{iv})$. If $\mathtt{decom}_I = \perp$, increment $\mathtt{ctr}$ and repeat steps 7-8.

9: The signature is $\sigma := ((\mathbf{c})_{i\in[\tau-1]}, \tilde{\mathbf{u}}, \mathbf{d}, (\tilde{a}_j)_{j\in[m-1]}, \mathtt{decom}_I, \mathtt{chl}_3, \mathtt{iv}, \mathtt{ctr})$.

Algorithm 5 Protocol $\mathsf{Verify}_{\mathsf{VOLE-LEP}}(\sigma, \mathtt{m}, \mathtt{pk})$

Input:

- the signature $\sigma := ((\mathbf{c})_{i\in[\tau-1]}, \tilde{\mathbf{u}}, \mathbf{d}, (\tilde{a}_j)_{j\in[m-1]}, \mathtt{decom}_I, \mathtt{chl}_3, \mathtt{iv}, \mathtt{ctr})$;
- the message $\mathtt{m}$ to be signed
- the public key $\mathtt{pk} = (\mathbf{G}_1, \mathbf{H}_2)$.

Verify:

1: Compute the system of degree-m equations $\{g_1, ..., g_t\} \in \mathbb{F}_p[X_1, ..., X_\ell]$ from $\mathtt{pk}$.
2: Compute $I = (\Delta_0, ..., \Delta_{\tau-1}) = \mathsf{GetChallenge}(\mathtt{chl}_3)$, $I \in \mathbb{F}_p^\tau$, and recover the VOLE correlation $(\overline{\mathtt{com}}, \mathbf{R}) \leftarrow \mathsf{VOLE.Reconstruct}(I, \mathtt{decom}_I, \mathbf{c}_1, ..., \mathbf{c}_{\tau-1}, \mathtt{iv}, \ell + m\tau)$.
3: Set $\overline{\mathtt{chl}}_1 \leftarrow \mathsf{H}(\mathtt{pk}||\overline{\mathtt{com}}||\mathbf{c}_1||...||\mathbf{c}_{\tau-1})$.
4: Run the VOLE consistency check of [17] with challenge $\mathtt{chl}_1$, obtaining $\tilde{\mathbf{R}}$. Recompute $\tilde{\mathbf{V}} = \tilde{\mathbf{R}} - \tilde{\mathbf{u}} \cdot [1 \mid \cdots \mid 1] \cdot \mathrm{diag}(\boldsymbol{\Delta})$.
5: Run the protocol from Model i, where the challenges are computed from $\overline{\mathtt{chl}}_2 = \mathsf{H}(\mathtt{chl}_1||\tilde{\mathbf{u}}||\tilde{\mathbf{V}}||\mathbf{d})$, obtaining the value $\tilde{c} \in \mathbb{F}_{p^\tau}$. Recompute

$$\tilde{a}_0^* = \tilde{c} - \sum_{k=1}^{m-1} \tilde{a}_k (\Delta^*)^k,$$

where Δ^* is the embedment of $(\Delta_0, ..., \Delta_{\tau-1})$ in $\mathbb{F}_{p^\tau}$.
6: Compute $\overline{\mathtt{chl}}_3 := \mathsf{H}(\overline{\mathtt{chl}}_2||\tilde{a}_0^*||\tilde{a}_1||\ldots||\tilde{a}_{m-1}||\mathtt{ctr})$.
7: Accept the signature if $\overline{\mathtt{chl}}_3 = \mathtt{chl}_3$. Otherwise, reject.

Remark 11 (Field element conversion) In Algorithm 4 and Algorithm 5, the procedure $\mathsf{GetChallenge}$ is used to convert the string of bits $\mathtt{chl}_3$ into a vector $I = (\Delta_0, \ldots, \Delta_{\tau-1}) \in \mathbb{F}_p^\tau$. Since in this work $p = 127$ and $\tau = 19$, $\mathtt{chl}_3$ is a string of $\tau \cdot \log_2(p) = 19 \cdot 7$ bits: then, each 7-bit chunk of $\mathtt{chl}_3$ is converted into its corresponding integer $\Delta_i, i \in [0, \tau - 1]$. To avoid the event $\Delta_i = 127$ for some $i \in [\tau]$ (which would not correspond to an element of $\mathbb{F}_{127}$), there are various options. The most immediate one is to rely on a rejection mechanism that rejects the invalid challenges and produce new one using a counter. The counter is then included in the signature so that the verifier can check that the challenge computation is correct. the procedure $\mathsf{GetChallenge}$ is defined as follows:

Algorithm 6 $\mathsf{GetChallenge}(\mathtt{chl}_3)$

1: **for** $i \in [0, \tau - 1]$ **do**
2: $\quad \Delta_i \leftarrow \mathsf{Int}(\mathtt{chl}_3[i, \ldots, i + 6], 2)$
3: $\quad$ **if** $\Delta_i = 127$ **then**
4: $\quad\quad$ **return** $\perp$
5: **return** $(\Delta_0, \ldots, \Delta_{\tau-1})$

Model	Witness length ℓ	Degree m
Model 1	$n^2 + n$	2
Model 2	$n^2 \cdot (\log_p(n) + 2) + n$	2
Model 3	$n \cdot d + n + n$	$2d + 1$

Table 2: Witness length ℓ and the degree m for the model from Section 4.

In general, $\Pr[\texttt{chl}_3 \text{ is rejected}] = 1 - \left(\frac{126}{127}\right)^{19} \approx 15\%$, thus the expected number of rejections is low.

5.1 Security of the signatures

Regarding the security of the signatures described in this section, the signatures derived from Model 1 and Model 2 benefit from the following result.

Proposition 9 *The signature scheme* $\mathsf{Sign}_{\mathsf{VOLE-LEP}}$ *is EUF-CMA secure.*

Proof The proof is a straightforward consequence of [3, Theorem 4] and Proposition 5 (or Proposition 7). Regarding Model 3, this is a consequence of [3, Appendix B.1] and Proposition 8). □

5.2 Performance Analysis

The VOLEitH framework is often used to reduce the size of the signature, as it avoids the usage of parallel repetitions. However, our signatures are much bigger than the original LESS signature due to the large sizes of our witnesses, which influence the lengths of the VOLE correlations. Indeed, the most substantial part of each signature consists of the corrections $\mathbf{c}_1, ..., \mathbf{c}_{\tau-1}$ of the VOLE secret $\mathbf{u}$ that the prover needs to send to the verifier. The signature sizes are computed according to the following formula

$$\text{Size} = \underbrace{\lambda}_{\texttt{iv}} + \underbrace{(\tau-1)\cdot\lceil\log_2(p)\rceil\cdot(\ell+m\tau)}_{\text{sVOLE correction vectors}} + \underbrace{(\ell+(m-1)\cdot\tau)\cdot\lceil\log_2(p)\rceil}_{\text{Masked witness + ZK output } (\tilde{a}_1,\ldots,\tilde{a}_{m-1})} + \\ + \underbrace{\lambda T_{\mathsf{open}} + 2\lambda\tau}_{\mathsf{pdecom}_I} + \underbrace{\tau\cdot\lceil\log_2(p)\rceil}_{\widetilde{u}} + \underbrace{\tau\cdot\lceil\log_2(p)\rceil}_{\texttt{chl}_3} + |\texttt{ctr}| \quad (31)$$

Table 2 reports the values of ℓ and m for our modellings.

Instead, a comparison of the signature sizes is provided in Table 3. We keep the same parameters as LESS for a more direct comparison, notice however that our signature size could benefit from a different choice of parameters, as detailed later in

Algorithm	Code Parameters			Size (kB)
LESS v2	$p = 127$	$n = 252$	$k = 126$	2.625
Model 1				1063.18
Model 2				4230.45
Model 3				49.877

Table 3: Signature sizes for $\lambda = 128$.

Remark 13.

Remark 12 The parameters displayed in Table 3, as well as the choice of $\tau = 19$, are kept unchanged for the three modellings. This choice was primarily motivated by the possibility to perform a direct comparison with LESS using the same parameters. Also, these parameters yield a negligible soundness error for both the degree-2 and the higher degree models. Indeed, thanks to [3, Theorem 4], we know that the protocol depicted in Algorithm 3 has soundness error $\frac{3}{p^\tau}$ which, with this choice of parameters, is smaller than 2^{-128}. Similarly, the higher degree protocol has soundness error [6]

$$\frac{1}{p^{r\tau}} + \left(1 - \frac{1}{p^{r\tau}}\right) \cdot \frac{d}{p^{r\tau}}$$

which, in turn, is smaller than 2^{-128} with our choice of parameters ($r = 1, \tau = 19$).

Even though our signatures are much longer than those achieved by LESS, our signature schemes achieve a lower computational complexity. Indeed, in LESS the verifier needs to perform Gaussian Elimination, which has a cost of roughly $O(n^3)$, while Model 2 and Model 3 have respectively $O(n^2 \log n)$ and $O(n^2 \log^2 n)$. Notice that however our models are defined over an degree-τ extension of $\mathbb{F}_p$ and not on the based field, as is it instead the case of LESS. Depending on the choice of the parameters, this could add an additional overhead up to $\tau^{\log_2(3)}$.

As for the security parameter λ, both our protocol and LESS would require to change the code parameters so that the LEP instances achieve the desired level of security. Then our protocol would require an higher τ, so that $2^\lambda \leq q^\tau$, while LESS would need more parallel repetitions t so that $2^\lambda \leq 2^t$.[1]

Remark 13 While our work targets LESS parameters to offer a more direct comparison, changing the parameters would realistically lead to better performance. In particular, working in characteristic 2 would require less VOLE correlations in Model 3 by removing the need of the squaring to do the XOR thus basically halving the degree of the constraints to $d + 1$

[1] To be more precise, since LESS use the fixed weight optimization, each instance of LESS change both the number of repetitions and the weight to reach a better balance between size and speed, making the proportionality factor harder to express with an explicit formula. Indeed the condition $2^\lambda \leq 2^t$ holds only for standard parallel repetition, while for fixed weight we should use $\binom{t}{w} \geq 2^\lambda$, thus two parameters can be tweaked to reach the desire balance. As now, since less use weights close to be $\frac{t}{2}$ doubling λ require to roughtly double t.

instead of $2d + 1$. This could also impact the field arithmetic, that is a major change from LESS. Finally, this could also change how the challenge are computed, possibly removing the need of the rejection sampling mechanism of Remark 11. However, doing so would require a complete new security analysis of the underlying LEP instance.

6 Conclusions

In this paper we presented three VOLE friendly modellings for LEP, that can be used to design efficient digital signature. Albeit larger than the original NIST submission, especially the two MQ based ones, our signature enjoys a lower computational complexity, making them interesting candidates for an alternative version of LESS, where signing time is more important than memory. While we were not able to directly implement and test our signature, the theoretical results are very promising, and we leave an the task of producing an efficient implementation as an open problem.

As a second contribution, we proposed two new reduction from LEP to MQ, that are interesting on their own. In particular, modeling a problem via a multivariate system of equation is the basis of many algebraic attacks, and thus our reductions can pave the way for new analysis on LEP. In this sense, we think that a deeper analysis of our obtained systems can be an interesting research directions. In particular it would be interesting to understand the behaviour of the Gröbner basis computation in both modelling and to get a theoretical estimate of the complexity to solve them. This is a difficult task, as it is often the case for very structured algebraic systems. Another interesting and natural follow-up to this work can be to analyse the impact of hybrid strategies on solving the proposed systems.

7 Acknowledgements

The second and third authors started this work at the University of Trento, and the core of this work was part of the second author's M.Sc. thesis. The second author is currently a PhD student at Marche Polytechnic University (Ancona, Italy), whereas the third author is currently affiliated with the Univerity of Bari Aldo Moro. The third author is a member of the INdAM Research Group GNSAGA. The first author is supported by the Italian Ministry of University and Research (MUR) under the PRIN 2022 program with projects "Mathematical Primitives for Post Quantum Digital Signatures" (CUP I53D23006580001) and "Post quantum Identification and eNcryption primiTives: dEsign and Realization (POINTER)" (CUP I53D23003670006). The first and second authors are supported by MUR under the Italian Fund for Applied Science (FISA 2022), project "Quantum-safe cryptographic tools for the protection of national data and information technology assets" (QSAFEIT) - No. FISA 2022-00618 (CUP I33C24000520001). The third author was partially supported by the project PRIN 2022SC, title "Algebraic Methods in Cryptanalysis", Grant Ref. 2022RFAZCJ, CUP H53C24000830006.

Appendix A Vector Commitment Schemes from GGM Trees

In this section, we show how to design a random vector commitment with all-but-one openings and a random vector commitment scheme with all-but-τ openings.

A.1 All-but-one Vector Commitment Scheme

Informally, a *random vector commitment scheme* (VC) allows a prover $\mathcal{P}$ to commit to a vector of random seeds while keeping it secret. In particular, in this section we define a vector commitment scheme with all-but-one openings: the prover commits to N random seeds, of which $N-1$ are later opened to the verifier. In this construction, the seeds are generated using a GGM tree of length-doubling pseudo-random generators (PRGs). This VC was originally defined in [3] and later used in the first version of FAEST ([17]).

Let $\mathsf{PRG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a length-doubling PRG, let $\mathsf{H}_0 : \{0,1\}^* \to \{0,1\}^{2\lambda}$ be a collision-resistant hash function, and let $\mathsf{H}_1 : \{0,1\}^\lambda \to \{0,1\}^\lambda \times \{0,1\}^{2\lambda}$ be a PRG and collision-resistant hash function. Finally, let $N = 2^d$. The random vector commitment VC is defined in Algorithm 7. From Algorithm 7, it follows that the random vector commitment VC is perfectly correct. Moreover, it can be shown that VC is also extractable binding and selectively hiding: the proofs of these statements can be found in [3, §3.1].

A.2 All-but-τ Vector Commitment Scheme

As mentioned in Section 3.3.1, VOLE-in-the-Head signatures use multiple instances of VC. However, this technique is quite inefficient, since it implies a decommitment size of $\lambda \log_2(N)$ per vector commitment, with N being the number of leaves in the tree. This has proven to be a large part of the final signature size. This cost can be improved by employing a *batch all-but-one vector commitment scheme* (BAVC). Introduced in [18], a batch vector commitment scheme is defined as follows.

Definition 10 (BAVC [18, Definition 2]) Let H be a random oracle. A *(non-interactive) batch all-but-one vector commitment scheme* BAVC With message space $\mathcal{M}$ in the random oracle model is defined by the following PPT algorithms, where all ofthem have access to a random oracle and obtain the security parameter 1^λ as well as $\tau, N_1, ..., N_\tau$ as input:

- $\mathsf{Commit}() \to (\mathtt{com}, \mathtt{decom}, (m_1^{(\alpha)}, ..., m_{N_\alpha}^{(\alpha)})_{\alpha\in[\tau]})$: Output a commitment $\mathtt{com}$ with opening information $\mathtt{decom}$ for messages $(m_1^{(\alpha)}, ..., m_{N_\alpha}^{(\alpha)})_{\alpha\in[\tau]}) \in \mathcal{M}^{N_1+...+N_\tau}$;
- $\mathsf{Open}(\mathtt{decom}, I) \to \mathtt{decom}_I \vee \perp$: On input an opening $\mathtt{decom}$ and the index vector $I \subset [N_1] \times \cdots \times [N_\tau]$, output $\perp$ or an opening $\mathtt{decom}_I$ for I;
- $\mathsf{Verify}(\mathtt{com}, \mathtt{decom}_I, I) \to ((m_j^{(\alpha)})_{j\in[N_\alpha]\setminus i_\alpha})_{\alpha\in[\tau]} \vee \perp$: Given a commitment $\mathtt{com}$, an opening $\mathtt{decom}_I$ for an index vector I, and the index vector I, either output all messages $(m_j^{(\alpha)})_{j\in[N_\alpha]\setminus i_\alpha}$ (accept the opening) or $\perp$ (reject the opening).

Algorithm 7 All-but-one vector commitment ([17, §5.1, Figure 5.1])

$\mathsf{VC.Commit}(\mathbf{r} \in \{0,1\}^\lambda, N = 2^d)$

$k_0^0 \leftarrow \mathbf{r}$
for $i \in [d]$ **do**
  for $j \in [0, 2^{i-1} - 1]$ **do**
    $(k_{2j}^i, k_{2j+1}^i) \leftarrow \mathsf{PRG}(k_j^{i-1})$
  end for
end for
for $j \in [0, N-1]$ **do**
  $(\mathtt{sd}_j, \overline{\mathtt{com}}_j) \leftarrow \mathsf{H}_1(k_j^d)$
end for
$h \leftarrow \mathsf{H}_0(\overline{\mathtt{com}}_0, ..., \overline{\mathtt{com}}_{N-1})$
$\mathtt{decom} \leftarrow$
  $(((k_j^i)_{j \in [0,2^i-1]})_{i \in [d]}, (\overline{\mathtt{com}}_j)_{j \in [0,N-1]})$
return $(\mathtt{com} := h, \mathtt{decom}, (\mathtt{sd}_j)_{j \in [0,N-1]})$

$\mathsf{VC.Open}(\mathtt{decom}, j^* \in [0, n-1])$

Write $j^* = \sum_{i=0}^{d-1} 2^i b_i$
Parse $\mathtt{decom} =$
  $(((k_j^i)_{j \in [0,2^i-1]})_{i \in [d]}, (\overline{\mathtt{com}}_j)_{j \in [0,N-1]})$
$a \leftarrow 0$
for $i \in [d]$ **do**
  $\mathsf{cop}[i] \leftarrow k_{2a+\overline{b_{d-i}}}$
  $a \leftarrow 2a + b_{d-i}$
end for
$\mathsf{pdecom} \leftarrow (\mathsf{cop}, \overline{\mathtt{com}}_{j^*})$
return pdecom

$\mathsf{VC.Reconstruct}(\mathsf{pdecom}, j^* = \sum_{i=0}^{d-1} 2^i b_i)$

Parse $\mathsf{pdecom} = (\mathsf{cop}, \overline{\mathtt{com}}_{j^*})$
$a \leftarrow 0, k_0^0 \leftarrow \perp$
for $i \in [d]$ **do**
  $k_{2a+\overline{b_{d-i}}} \leftarrow \mathsf{cop}[i]$
  $k_{2a+b_{d-i}} \leftarrow \perp$
  for $j \in [0, 2^{i-1} - i] \backslash \{a\}$ **do**
    $(k_{2j}^i, k_{2j+1}^i) \leftarrow \mathsf{PRG}(k_j^{i-1})$
  end for
  $a \leftarrow 2a + b_{d-i}$
end for
for $j \in [0, N-1] \backslash \{j^*\}$ **do**
  $(\mathtt{sd}_j, \overline{\mathtt{com}}_j) \leftarrow \mathsf{H}_1(k_j^d)$
end for
$h \leftarrow \mathsf{H}_0(\overline{\mathtt{com}}_0, ...\overline{\mathtt{com}}_{N-1})$
return $(h, (\mathtt{sd}_j)_{j \in [0,N-1] \backslash \{j^*\}})$

$\mathsf{VC.Verify}(\mathtt{com}, \mathsf{decom}, j^* = \sum_{i=0}^{d-1} 2^i b_i)$

$(\overline{\mathtt{com}}, (\mathtt{sd}_j)_{j \in [0,N-1] \backslash \{j^*\}}) \leftarrow$
  $\mathsf{VC.Reconstruct}(\mathsf{pdecom}, j^*)$
if $\mathtt{com} = \overline{\mathtt{com}}$ **then**
  return 1
else
  return 0
end if

As previously mentioned, we can express τ single VCs as one BAVC. The idea is to generate a single GGM tree with $\sum_{\alpha=1}^{\tau} N_\alpha$ leaves and open τ of them. The first τ leaves correspond to the first entry of the τ vectors, the second τ to the second entry, and so on. In this way, the active paths of the tree will merge with high probability, thus reducing the number of internal nodes to be revealed. However, this number still depends on the index vector I. Since this could cause some fluctuation in the size of the final signature, a threshold T_{open} is fixed to bound the number of internal nodes to be opened. Given T_{open}, the open algorithm aborts whenever T_{open} is exceeded. In the description of the BAVC, we use the following elements:

- a pseudo-random generator (PRG) $\mathsf{PRG} : \{0,1\}^\lambda \times \{0,1\}^{128} \to \{0,1\}^{2\lambda}$;
- a collision-resistan hash function (CRHF) $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$;
- a PRG and CRHF $\mathsf{G} : \{0,1\}^\lambda \times \{0,1\}^{128} \to \{0,1\}^\lambda \times \{0,1\}^{2\lambda}$;
- the number of vectors τ;
- the lengths of the vectors $N_1, ..., N_\tau$;
- the opening threshold T_{open};
- $\pi : [L-1, 2L-2] \to \{(\alpha, i)\}_{1 \le i \le N_\alpha}$ a bijective mapping from the leaves of the GGM tree to the positions in the vector commitments.

The BAVC is defined in Algorithm 8. The definition of the algorithms is adapted from [18] and [2]. From Algorithm 8 it follows that this BAVC is perfectly correct, and it can be proved that it is extractable binding and selectively hiding. The definitions of these properties for a BAVC are found in [18, §3], while the respective proofs can be found in [18, Lemma 6, Lemma 7].

Appendix B Generating VOLE Correlations

We report here the algorithms used to generate reconstruct VOLE correlations. We focus on the generation procedure exploiting a BAVC, as done in the second version of FAEST [2], instead of τ all-but-one vector commitments scheme, as in the first version of FAEST [17]. We adapt the algorithms defined in [2, §5.5] to our framework, where we need to generate $\tau = \lceil \lambda / \log_2(p) \rceil$ VOLE correlations over $\mathbb{F}_p$ and adjust them using subspace VOLE to obtain a single correlation over $\mathbb{F}_{p^\tau}$: for the sake of comprehension, we use the same names for the algorithms as in [2]. Hence, hereafter $N_0 = \cdots = N_{\tau-1} = N = 2^{\lceil \log_2(p) \rceil}$, and the single VOLE correlations will be computed directly over $\mathbb{F}_p$ instead of over an extension. As a last bit of notation, in what follows we let PRG_p be a pseudorandom generator that, given as input a seed $\mathtt{sd} \in \{0,1\}^*$, an initialization vector $\mathtt{iv} \in \{0,1\}^{128}$ and a positive integer $\hat{\ell} \in \mathbb{N}$, return a vector of the specified length $\hat{\ell}$ over $\mathbb{F}_p$. the procedure is defined in Algorithm 9.

Algorithm 8 Batch all-but-one vector commitment

$\mathsf{BAVC.Commit}(r \in \{0,1\}^\lambda, \mathtt{iv} \in \{0,1\}^{128})$

$k_0 \leftarrow r$
for $i \in [0, L-2]$ **do**
  $(k_{2i+1}, k_{2i+2}) \leftarrow \mathsf{PRG}(k_i, \mathtt{iv})$
$\{\mathtt{sd}_1^{(\alpha)}, ..., \mathtt{sd}_{N_\alpha}^{(\alpha)}\}_{\alpha \in [\tau]} \xleftarrow{\pi} \{k_{L-1}, .., k_{2L-2}\}$
for $\alpha \in [\tau]$ **do**
  for $i \in [N_\alpha]$ **do**
    $(m_i^{(\alpha)}, \mathtt{com}_i^{(\alpha)}) \leftarrow \mathsf{G}(\mathtt{sd}_i^{(\alpha)}, \mathtt{iv})$
for $\alpha \in [\tau]$ **do**
  $h^{(\alpha)} \leftarrow \mathsf{H}(\mathtt{com}_1^{(\alpha)}, ..., \mathtt{com}_{N_\alpha}^{(\alpha)})$
$\mathtt{com} = h \leftarrow \mathsf{H}(h^{(1)}, ..., h^{(\tau)})$
$\mathtt{decom} \leftarrow k_0$
return $\left(\mathtt{com}, \mathtt{decom}, (m_i^{(\alpha)})_{i \in [N_\alpha], \alpha \in [\tau]}\right)$

$\mathsf{BAVC.Open}(\mathtt{decom}, I = (i^{(1)}, ..., i^{(\tau)}), \mathtt{iv})$

Recompute k_j for $j \in [0, 2L-2]$ from k_0 as in $\mathsf{Commit}(r, \mathtt{iv})$
$S = \{k_{L-1}, ..., k_{2L-2}\}$
$\forall \alpha \in [\tau]$, remove $k_{\pi^{-1}(\alpha, i_\alpha)}$ from S
for $i \in [L-2, 0]$ **do**
  if $k_{2i+1} \in S \wedge k_{2i+2} \in S$ **then**
    Replace them in S with k_i
if $|S| < T_{\mathsf{open}}$ **then**
  return $\mathtt{decom}_I = \left((\mathtt{com}_{i^{(\alpha)}}^{(\alpha)})_{\alpha \in [\tau]}, S\right)$
else return $\perp$

$\mathsf{BAVC.Reconstruct}(\mathtt{com} = h, \mathtt{decom}_I = ((\mathtt{com}_{i^{(\alpha)}}^{(\alpha)})_{\alpha \in [\tau]}, S), I = (i^{(1)}, ..., i^{(\tau)}), \mathtt{iv})$

Recompute $\mathtt{sd}_i^{(\alpha)}$ from $\mathtt{decom}_I$ for $\alpha \in [\tau]$ and $i \neq i^{(\alpha)}$ using the internal nodes in S
Compute $(m_i^{(\alpha)}, \mathtt{com}_i^{(\alpha)}) \leftarrow \mathsf{G}(\mathtt{sd}_i^{(\alpha)}, \mathtt{iv})$ for $\alpha \in [\tau]$ and $i \neq i^{(\alpha)}$
for $\alpha \in [\tau]$ **do**
  $h^{(\alpha)} \leftarrow \mathsf{H}(\mathtt{com}_1^{(\alpha)}, ..., \mathtt{com}_{N_\alpha}^{(\alpha)})$
if $h \neq \mathsf{H}(h^{(1)}, ..., h^{(\tau)})$ **then return** $\perp$
return $\left(\mathtt{com}, (m_i^{(\alpha)})_{i \in [N_\alpha] \setminus \{i^{(\alpha)}\}}\right)_{\alpha \in [\tau]}$

Algorithm 9 VOLE-in-the-Head functions

$\mathsf{P.ConvertToVOLE}(\mathtt{sd}_0, ..., \mathtt{sd}_{N-1}, \mathtt{iv}, \hat{\ell})$

for $j \in [0, p-1]$ **do**
  $\mathbf{t}_j \leftarrow \mathsf{PRG}_p(\mathtt{sd}_j, \mathtt{iv}, \hat{\ell})$
$\mathbf{u} \leftarrow \sum_{j=0}^{p-1} \mathbf{t}_j$
$\mathbf{v} \leftarrow -\sum_{j=0}^{p-1} j \cdot \mathbf{t}_j$
return $(\mathbf{u}, \mathbf{v})$

$\mathsf{V.ConvertToVOLE}(\Delta, (\mathtt{sd}_j)_{j \in [0,N-1] \setminus \{\Delta\}}, \mathtt{iv}, \hat{\ell})$

for $j \in [0, p-1] \setminus \{\Delta\}$ **do**
  $\mathbf{t}_j \leftarrow \mathsf{PRG}_p(\mathtt{sd}_j, \mathtt{iv}, \hat{\ell})$
$\mathbf{r} \leftarrow \sum_{j=0}^{p-1} (\Delta - j) \cdot \mathbf{t}_j$
return $\mathbf{r}$

$\mathsf{VOLE.Commit}(\mathbf{r}, \mathtt{iv}, \hat{\ell})$

$(\mathtt{com}, \mathtt{decom}, (\mathtt{sd}_0^i, ..., \mathtt{sd}_{N_i-1}^i)_{i \in [0,\tau-1]}) \leftarrow \mathsf{BAVC.Commit}(\mathbf{r})$
for $i \in [0, \tau-1]$ **do**
  $(\mathbf{u}_i, \mathbf{v}_i) \leftarrow \mathsf{P.ConvertToVOLE}(\mathtt{sd}_0^i, ..., \mathtt{sd}_{N_i-1}^i, \mathtt{iv}, \hat{\ell})$
$\mathbf{V} \leftarrow [\mathbf{v}_0 \mid \cdots \mid \mathbf{v}_{\tau-1}]$
$\mathbf{u} \leftarrow \mathbf{u}_0$
for $i \in [0, \tau-1]$ **do**
  $\mathbf{c}_i \leftarrow \mathbf{u}_i - \mathbf{u}$
return $(\mathtt{com}, \mathtt{decom}, \mathbf{c}_1, ..., \mathbf{c}_{\tau-1}, \mathbf{u}, \mathbf{V})$

$\mathsf{VOLE.Reconstruct}(I, \mathtt{decom}_I, \mathbf{c}_1, ..., \mathbf{c}_{\tau-1}, \mathtt{iv}, \hat{\ell})$

if $I = \perp$ **then return** $\perp$
else do
  $\mathsf{out} := \left(\mathtt{com}, (\mathtt{sd}_j^i)_{j \in [0,N_i-1] \setminus \{\Delta_i\}, i \in [0,\tau-1]}\right) \leftarrow \mathsf{BAVC.Reconstruct}(\mathtt{decom}_I, I, \mathtt{iv})$
  if $\mathsf{out} = \perp$ **then return** $\perp$
  else do
    for $i \in [0, \tau-1]$ **do**
      $\mathbf{r}'_i \leftarrow \mathsf{V.ConvertToVOLE}((\mathtt{sd}_j^i)_{j \in [0,N_i-1] \setminus \{\Delta_i\}}, \mathtt{iv}, \hat{\ell})$
      $\mathbf{r}_i \leftarrow \mathbf{r}'_i - \mathbf{c}_i \cdot \Delta_i$
    $\mathbf{R} = [\mathbf{r}_0 \mid \cdots \mid \mathbf{r}_{\tau-1}]$
    return $(\mathtt{com}, \mathbf{R})$

Appendix C Alternative description of Model 3

In this section, we provide an alternative description of Model 3, more similar to the original work of [4].

Recall that, thanks to Proposition 2, we aim to prove the knowledge of a monomial matrix $\mathbf{Q} \in \mathsf{M}_n(p)$ satisfying Equation (11). Observe that a monomial matrix can be split into the product of a diagonal matrix $\mathbf{D}$ and a permutation matrix $\mathbf{P}$, i.e. $\mathbf{Q} = \mathbf{DP}$. Hence, we divide the design in three steps: firstly, we use the algorithms defined in [4] to commit to $\mathbf{P}$, then we separately commit to the nonzero entries of $\mathbf{D}$ and, lastly, we prove that $\mathbf{Q} = \mathbf{DP}$ satisfies Equation (11) via an arithmetic circuit. In the description of this model, we will use the following notation.

Notation For shortness, we will write $d = \lceil \log_2(n) \rceil$. We will denote $\mathcal{B}^j \in \{0,1\}^d$ the binary representation of an integer $j \in [0, n-1]$, and by $\overline{\mathcal{B}^j}$ its complement. In this model, we work with VOLE correlations of length 1 and denote them using polynomials: the prover $\mathcal{P}$ learns a polynomial $f_u(X) = u \cdot X + v$, while the verifier learns Δ and $r_u = f_u(\Delta)$. Notice that the committed value is stored in the leading coefficient of f_u. Moreover, the following algorithms exploit *higher degree commitments*, which are obtained from the multiplication of two or more standard VOLE correlations: as usual, the committed value is stored in the leading coefficient of the resulting polynomial. To emphasize the degree of the commitment and the value being committed, we write $[\![u]\!]^{(m)} = f_u(X) = u \cdot X^m + a_{m-1} \cdot X^{m-1} + \cdots + a_1 \cdot X + a_0$ to represent a degree m commitment to u. The verifier's view of the commitment is still denoted as $r_u = f_u(\Delta)$. When $m = 1$, we simply write $[\![u]\!]^{(1)} = [\![u]\!]$.

Remark 14 In what follows, we will often need to show that a degree-m commitment opens to 0: in other words, suppose we know $[\![u]\!]^{(m)} = f_u(X)$ and we want to show that $u = 0$. Notice that it is equivalent to proving that the $\deg(f_u) < m$. This is achieved by the protocol described in Algorithm 10 and Algorithm 11. Similarly, we will often need to prove that a degree-m commitment opens to 1. To do so, given $[\![u]\!]^{(m)}$ we compute the polynomial

$$f(X) = f_u(X) - X^m$$

and use Algorithm 10 and Algorithm 11 to prove that $\deg(f) < m$.

Algorithm 10 $\mathsf{P.CheckZero}([\![u]\!]^{(m)}, ([\![s_i]\!])_{i \in [0,m-2]})$ [4, Algorithm 2.1]

Public parameters: Degree of input VOLE correlation m

Prover's input: Degree-m VOLE correlation $[\![u]\!]^{(m)} = f_u(X)$, $(m-1)$ random VOLE correlations $[\![s_i]\!] = f_{s_i}(X) = s_i X + \sigma_i$, where $s_i, \sigma_i \in \mathbb{F}_{p^\tau}$ for $i \in [0, m-2]$

Prover's output: VOLE correlation $[\![A]\!]^{(m)}$

1: $f_{\mathsf{mask}}(X) = \sum_{i=0}^{m-2} f_{s_i}(X) \cdot X^i$
2: $[\![A]\!]^{(m)} = f_A(X) = f_u(X) + f_{\mathsf{mask}}(X)$
3: **return** $[\![A]\!]^{(m)}$

Algorithm 11 V.CheckZero($[\![A]\!]^{(m)}$, r_u, $(r_{s_i})_{i\in[0,m-2]}$, Δ) [4, Algorithm 2.2]

Public parameters: Degree of input VOLE correlation m

Verifier's input: Prover's output $[\![A]\!]^{(m)}$, $r_u = f_u(\Delta), r_{s_i} = f_{s_i}(\Delta)$ for $i \in [0, m-2]$, Δ

Verifier's output: Boolean B indicating if the leading coefficient of f_A is equal to zero

1: $r = r_u + \sum_{i=0}^{m-2} r_{s_i} \cdot \Delta^i$
2: $B \leftarrow (r \stackrel{?}{=} f_A(\Delta))$
3: **return** B

Remark 15 In Algorithm 10 and Algorithm 11, for the masking to be successful, the correlations $([\![s_i]\!])_{i\in[0,m-2]}$ need to be constructed such that $s_i \in \mathbb{F}_{p^\tau}$ (normally, we would have $s_i \in \mathbb{F}_p$). As described in [18, Appendix A.3], these VOLE correlations are generated by concatenating τ "standard "VOLEs over $\mathbb{F}_p$ and viewing the rows of the resulting matrices as elements of $\mathbb{F}_{p^\tau}$.

We start by reporting the techniques from [4]. In the work of Bettaieb et al., the permutation matrix $\mathbf{P}$ is represented by n integers $\mathsf{pos}_0, ..., \mathsf{pos}_{n-1} \in [0, n-1]$, which correspond to the positions of the non-zero entries in each row of $\mathbf{P}$. Then, random VOLE correlations are used to reconstruct each row of $\mathbf{P}$, by computing one commitment for each element of the rows. This procedure, adapted to our use case, is described in Algorithm 12 and Algorithm 13.

As said before, given two binary $\mathbf{b}^1, \mathbf{b}^2$, $\mathbf{b}^1 \oplus \mathbf{b}^2 = (1, ..., 1)^\top$ if and only if $\mathbf{b}^2 = \overline{\mathbf{b}^1}$. As a consequence, the product of the entries of $\mathbf{b}^1 \oplus \mathbf{b}^2$ is equal to 1 when $\mathbf{b}^2 = \overline{\mathbf{b}^1}$ and 0 otherwise. Given a integer $\mathsf{pos} \in [0, n-1]$, let $\mathbf{p} = \mathcal{B}^{\mathsf{pos}} \in \mathbb{F}_2^d$. Then, the vector $\mathbf{z} \in \mathbb{F}_2^n$ defined as

$$\forall j \in [0, n-1].\ z_j := \prod_{k=0}^{d-1} (\overline{\mathcal{B}^j}_k \oplus p_k)$$

contains exactly one entry equal to 1 a position $j = \mathsf{pos}$, while all its other coordinates are equal to 0. In other words, this procedure reconstructs an elementary vector z given the position of the 1. In [4], this procedure is used to construct degree-d VOLE correlations $[\![\mathbf{z}]\!]^{(d)}$ for an elementary vector in $\mathbb{F}_2^n$ from a secret position pos.

Unfortunately, we cannot directly apply their modeling to our case, since we are working on $\mathbb{F}_p$ instead of $\mathbb{F}_2$. Indeed, we have two problems: first, we have no guarantee that the vector are binary, while this come for free wehn working in $\mathbb{F}_2$ we need to add additional checks to force the prover to only use binary vectors in $\mathbb{F}_p$. Second, when working in odd characteristic $-1 \neq 1$, thus to implement the XOR we need an higher degree relations. Indeed, given $\mathsf{pos} \in [0, n-1]$, let $\mathbf{p} = \mathcal{B}^{\mathsf{pos}} \in \mathbb{F}_p^d$ and define its complement $\overline{\mathcal{B}^{\mathsf{pos}}}$ to be the vector such that $\mathbf{p} + \overline{\mathcal{B}^{\mathsf{pos}}} = (1, \ldots, 1)^\top \in \mathbb{F}_p^d$. Then, in this case,

$$\prod_{k=0}^{d-1} (p_k - \overline{\mathcal{B}^{\mathsf{j}}}_k) = \begin{cases} \pm 1 \text{ if } j = \mathsf{pos} \\ 0 \text{ otherwise} \end{cases}$$

Algorithm 12 P.VOLE-ElementaryVector($\mathsf{pos}, (\llbracket u_k \rrbracket)_{k\in[0,d-1]}$) (adapted from [4, Algorithm 3.1])

Public parameters: Length of secret vector n, $d = \lceil \log_2(n) \rceil$.
Prover's input: Secret position $\mathsf{pos} \in [0, n-1]$, d random VOLE correlations $\llbracket u_k \rrbracket = f_{u_k}(X) = u_k X + v_k$, with $u_k \in \mathbb{F}_p, v_k \in \mathbb{F}_{p^\tau}$ for $k \in [0, d-1]$.
Prover's output: Masked secret $\mathbf{t} \in \mathbb{F}_p^d$, degree-$d$ VOLE correlations $\llbracket \mathbf{z} \rrbracket^{(d)}$, degree-2 VOLE correlations $\llbracket \mathsf{PosCheck} \rrbracket^{(2)}$

Compute masked secret
1: $\mathbf{p} = \mathcal{B}^{\mathsf{pos}}$
2: $\mathbf{t} = \mathbf{p} - \mathbf{u}$, where $\mathbf{u} = (u_0, ..., u_{d-1})$
Compute $\mathbf{p}$'s checks
3: **for** $k \in [0, d-1]$ **do**
4: $\quad \llbracket p_k \rrbracket = f_{p_k}(X) = p_k X + v_k$
5: $\quad \llbracket \mathsf{PosCheck}_k \rrbracket^{(2)} = (f_{p_k}(X) - X) \cdot f_{p_k}(X)$
6: $\boldsymbol{\alpha} \xleftarrow{\$} (\mathbb{F}^*_{p^\tau})^d$
7: $\llbracket \mathsf{PosCheck} \rrbracket^{(2)} = \sum_{k=0}^{d-1} \alpha_k \llbracket \mathsf{PosCheck}_k \rrbracket^{(2)}$
Compute elementary vector
8: **for** $j \in [0, n-1]$ **do**
9: $\quad$ **for** $k \in [0, d-1]$ **do**
10: $\quad\quad \beta_k = p_k - \overline{\mathcal{B}^j}_k$
11: $\quad\quad f_{\beta_k}(X) = \beta_k X + v_k$
12: $\quad \llbracket z_j \rrbracket^{(2d)} = \left(\prod_{k=0}^{d-1} f_{\beta_k}(X) \right)^2$
Compute the output and return
13: $\llbracket \mathbf{z} \rrbracket^{(2d)} = (\llbracket z_0 \rrbracket^{(2d)}, ..., \llbracket z_{n-1} \rrbracket^{(2d)})$
14: **return** $(\mathbf{t}, \llbracket \mathbf{z} \rrbracket^{(2d)}, \llbracket \mathsf{PosCheck} \rrbracket^{(2)})$

Hence, to reconstruct the elementary vector $\mathbf{z} \in \mathbb{F}_p^n$ containing a 1 at position $j = \mathsf{pos}$, its entries must be defined as

$$\forall j \in [0, n-1].\ z_j := \prod_{k=0}^{d-1} (p_k - \overline{\mathcal{B}^j}_k)^2$$

With this adjustement, this procedure can be used to to construct degree-$2d$ VOLE correlations $\llbracket \mathbf{z} \rrbracket^{(d)}$ for an elementary vector in $\mathbb{F}_p^n$ from a secret position pos, as depicted in Algorithm 12, lines 9-13. Indeed, the polynomial at Line 12 (Algorithm 12) will have maximum degree with leading coefficient 1 if and only if $j = \mathsf{pos}$ (i.e. $\mathcal{B}^j = \mathcal{B}^{\mathsf{pos}} = \mathbf{p}$). When $j \neq \mathsf{pos}$, $\llbracket z_j \rrbracket^{(2d)}$ will have degree smaller than $2d$, meaning that each $\llbracket z_j \rrbracket^{(2d)}$ will open to 0. Hence, by using Algorithm 12 n times (one per row of $\mathbf{P}$), we obtain n vectors $\{\llbracket \mathbf{z}_j \rrbracket^{(2d)}\}_{j\in[n]}$ which completely reconstruct $\mathbf{P}$ (when looking at the coefficients of the monomials of degree $2d$ of each polynomial). The same reasoning applies for the verifier's view of the protocols.

Algorithm 13 $\mathsf{V.VOLE\text{-}ElementaryVector}\left(\mathbf{t}, (r_{u_k})_{k\in[0,d-1]}, \Delta\right)$ (adapted from [4, Algorithm 3.2])

Public parameters: Length of secret vector n, $d = \lceil \log_2(n) \rceil$.

Verifier's input: Masked secret $\mathbf{t} \in \mathbb{F}_p^d$, $r_{u_k} = f_{u_k}(\Delta)$ for $k \in [0, d-1]$, Δ

Verifier's output: VOLE correlations $r_{\mathbf{z}}$, r_{PosCheck}

Compute VOLE correlations for $\mathbf{p}$'s check

1: **for** $k \in [0, k-1]$ **do**

2: $\quad r_{p_k} = t_k\Delta + r_{u_k}$

3: $\quad r_{\mathsf{PosCheck}_k} = (r_{p_k} - \Delta) \cdot r_{p_k}$

4: $\boldsymbol{\alpha} \xleftarrow{\$} (\mathbb{F}^*_{p^\tau})^d$

5: $r_{\mathsf{PosCheck}} = \sum_{k=0}^{d-1} \alpha_k r_{\mathsf{PosCheck}_k}$

Compute VOLE correlations for $\mathbf{z}$

6: **for** $j \in [0, n-1]$ **do**

7: $\quad$ **for** $k \in [0, d-1]$ **do**

8: $\quad\quad \gamma_k = t_k - \overline{\mathcal{B}^j}_k$

9: $\quad\quad r_{\beta_k} = r_{u_k} + \gamma_k\Delta$

10: $r_{z_j} = \left(\prod_{k=0}^{d-1} r_{\beta_k}\right)^2$

Compute the output and return

11: $r_{\mathbf{z}} = (r_{z_0}, ..., r_{z_{n-1}})$

12: **return** $(r_{\mathbf{z}}, r_{\mathsf{PosCheck}})$

Line 1-3 are necessary to ensure that $\mathbf{p}$ is a binary vector. The basic idea is that a vector $\mathbf{p}$ has all the entries in $\{0, 1\}$ if and only if the following condition holds:

$$\forall k \in [0, d-1],\ (p_k - 1) \cdot p_k = 0 \tag{C1}$$

i.e. they satisfy the set $\mathcal{S}_{\mathsf{pos}}$ of Section 4.3. We can exploit the VOLE correlations $[\![u_k]\!] = f_{u_k}(X) = u_k X + v_k$, with $u_k \in \mathbb{F}_p, v_k \in \mathbb{F}_{p^\tau}$ for $k \in [0, d-1]$, to define the polynomials

$$[\![p_k]\!] = f_{p_k}(X) = p_k X + v_k\ ,\ k \in [0, d-1].$$

Then, it can be easily seen that the polynomials

$$[\![\mathsf{PosCheck}_k]\!]^{(2)} = (f_{p_k}(X) - X) \cdot f_{p_k}(X)\ ,\ k \in [0, d-1] \tag{C2}$$

have degree 1 if and only if $\mathbf{p} \in \{0, 1\}^d$. Indeed

$$(f_{p_k}(X) - X) \cdot f_{p_k}(X) = (p_k X + v_k - X) \cdot (p_k X + v_k) = ((p_k - 1) \cdot p_k) X^2 + (2p_k v_k - v_k) X + v_k^2$$

and the first coefficient is 0 if and only if $p_k \in \{0,1\}$.

Then, to prove that $\mathbf{p} \in \{0,1\}^d$, the prover can use the CheckZero protocol to show that the polynomials in Equation (C2) have degree smaller than 2. Given the knowledge of the masked vector $\mathbf{t}$, the verifier can repeat this procedure by computing

$$r_{p_k} = t_k\Delta + r_{u_k} \ , \ \forall k \in [0, d-1]$$
$$r_{\mathsf{PosCheck}_k} = (r_{p_k} - \Delta) \cdot r_{p_k} \ , \ \forall k \in [0, d-1]$$

Remark 16 Notice that this procedure requires just 2 additional VOLE correlations for the CheckZero protocol. Indeed, Algorithm 10 can be run on a linear combination of the $[\![\mathsf{PosCheck}_k]\!]^{(2)}$ polynomials. This step is depicted in Algorithm 14.

Proposition 10 *Let $\mathbf{P} = [x_{i,j}]$ be a $n \times n$ matrix with coefficient in $\mathbb{F}_p$. Then each row of $\mathbf{P}$ has weight exactly one if and only if Algorithm 12 and Algorithm 13 return true.*

Sketch The proof is a straightworward application of the above reasoning. First, notice that if the prover does not cheat taking as input the positions directly implies that each row of $\mathbf{P}$ has exactly one non-zero entry. We are only left to show that the prover cannot cheat. This is a direct consequence of [4] with two differences:

- Line 1-3 force the prover to use a binary vector $\mathbf{p}$. Indeed, suppose that $\mathbf{p}$ is non binary, then $(f_{p_k}(X) - X) \cdot f_{p_k}(X)$ would have degree 2 for at least one position. Thus the check on $[\![\mathsf{PosCheck}]\!]^{(2)}$ would fail.
- Instead of computing the XOR of two vectors we compute their squared difference. It is immediate to see that given two binari vector $\mathbf{a}, \mathbf{b}$ $(a_i - b_i)^2 = 1$ if and only if $a_i - b_i = \pm 1$ (thus $a_i \neq b_i$) and $(a_i - b_i)^2 = 0$ if and only if $a_i - b_i = 0$ (thus $a_i = b_i$).

□

As said, the above protocol ensures that each row of $\mathbf{P}$ has weight exactly one. To show that $\mathbf{P}$ is a permutation matrix, we need to show that the same property holds for its columns (i.e. they satisfy the set $\mathcal{S}_{\mathsf{col}}$ of Section 4.3). To do so, we sum the commitments over each column of $\mathbf{P}$ and check that the resulting polynomials have leading coefficient equal to 1 as described in Remark 14. For this part, we use the same procedure as [4, Algorithm 3.5, Algorithm 3.6], except for the degree of the commitments. Hence, we will not report it fully, but it can be found in Line 1-5 of Algorithm 14.

To complete the protocol we now need to show that $\mathbf{D}$ is a diagonal matrices with no 0 entries. The key idea is to show that each diagonal entry has an inverse in $\mathbb{F}_p$ (i.e. they satisfy the set $\mathcal{S}_{\mathsf{const}}$ of Section 4.3). Since 0 is the only element without an inverse this a perfect characterization and does not leak anything.

Let $c_0, ..., c_{n-1} \in \mathbb{F}_p^*$ be such that $\mathbf{D} = \mathrm{diag}(c_0, ..., c_{n-1})$. The procedure is depicted in lines 6-12 of Algorithm 14. Firstly, the prover uses n random VOLE correlations

$[\![s_i]\!] = s_i X + \sigma_i$ to commit to the constants $c_0, ..., c_{n-1}$ by computing

$$e_i = c_i - s_i$$
$$[\![c_i]\!] = e_i X + [\![s_i]\!] = c_i X + \sigma_i$$

Then, to prove that $c_i \neq 0 \ \forall i \in [0, n-1]$, we employ n additional VOLE correlations $[\![\overline{s}_i]\!] = \overline{s}_i X + \overline{\sigma}_i$ to commit to $\overline{c}_i = c_i^{p-2}$. Let $[\![\overline{c}_i]\!]$ be the resulting commitment. Then, to show that $c_i \neq 0$, we compute

$$[\![\mathsf{ConstCheck}_i]\!]^{(2)} = [\![c_i]\!][\![\overline{c}_i]\!] - X^2 \tag{C3}$$

and use the CheckZero protocol to prove that $[\![\mathsf{ConstCheck}_i]\!]^{(2)}$ commits to 0. Indeed, if $c_i \neq 0$, then $\overline{c}_i \cdot c_i = 1$ in $\mathbb{F}_p$, and thus the degree of $[\![\mathsf{ConstCheck}_i]\!]^{(2)}$ will be equal to 1. Instead, whenever $c_i = 0$, $\overline{c}_i \cdot c_i = 0$ and $[\![\mathsf{ConstCheck}_i]\!]^{(2)}$ will fail the CheckZero verification. The prover then sends $\{e_i = c_i - s_i\}_{i\in[0,n-1]}$ and $\{\overline{e}_i = \overline{c}_i - \overline{s}_i\}_{i\in[0,n-1]}$ to the verifier, who can repeat the protocol using his parts of the VOLE correlations.

Remark 17 This approach does not comply exactly with the standard VOLE-based approach. Usually, the embedding of the witness into a VOLE correlations is done only once, meaning that the prover and the verifier share only one VOLE correlation for each piece of the witness. However, here the prover and the verifier share two sets of VOLE correlations for the constants $c_0, ..., c_{n-1}$ on the diagonal of $\mathbf{D}$, those committing to the constants and those committing to their inverses $\overline{c}_0, ..., \overline{c}_{n-1}$. This choice however does not reveal additional information to the verifier. Admittedly, for each $i \in [0, n-1]$, the verifier knows the following system of linear equations:

$$\begin{cases} c_i\Delta + \sigma_i = r_{c_i} \\ \overline{c}_i\Delta + \overline{\sigma}_i = r_{\overline{c}_i} \end{cases}$$

in the four unknowns $c_i, \overline{c}_i, \sigma_i, \overline{\sigma}_i$. However, the system is clearly underdetermined. Nonetheless, the verifier knows that $\overline{c}_i = c_i^{p-2}$, and can obtain the following equivalent system of two polynomial equations in three unknowns:

$$\begin{cases} c_i\Delta + \sigma_i = r_{c_i} \\ c_i^{p-2}\Delta + \overline{\sigma}_i = r_{\overline{c}_i} \end{cases}$$

This system is still underdetermined, but has one degree of freedom instead of two. This reduces the number of possible solutions, but an apt choice of the parameter p still makes it infeasible for the verifier to learn the true value of c_i. Moreover, recall that in order to recover the whole transformation, the verifier should solve n such systems, one for each constant c_i, $i \in [0, n-1]$. For completeness, we mention that an alternative approach to check that $c_i \neq 0$ can be obtained by computing

$$[\![\mathsf{ConstCheck}_i]\!]^{(p-1)} = ([\![c_i]\!])^{p-1} - X^{p-1} \ , \ i \in [0, n-1]$$

instead of the polynomial in Equation (C3), and check that $\deg([\![\mathsf{ConstCheck}_i]\!]^{(p-1)}) < p-1$ as described in Remark 14. This technique requires less VOLE correlations, since we are no more committing to the inverse of c_i. However, it increases the communication cost, as the size of the output of Algorithm 10 depends on the degree of its input.

Proposition 11 *Let* $\mathbf{D}$ *be a matrix. Then* $\mathbf{D}$ *satisfy the above conditions if and only if* $\mathbf{D}$ *its diagonal elements are non-zero.*

Proof If all the diagonal elements of $\mathbf{D}$ are non zero then the conditions are trivially satisfied. On the other hand, let $\mathbf{D}$ be a matrix that satisfy the above condition. That means that every diagonal element has an inverse and hence is non zero □

The above conditions only care about the diagonal elements, thus in principle $\mathbf{D}$ could also be a non-diagonal matrix. To check this we show that $\mathbf{Q} = \mathbf{DP}$ satisfies Equation (11).

Indeed, similarly to what has been done in [4], we reconstruct the product $\mathbf{G}_1\mathbf{DPH}_2^\top$ using the commitments to $\mathbf{P}$ and $\mathbf{D}$ as shown in lines 13-20 of Algorithm 14. Since the commitment to $\mathbf{D}$ is simply a commitment to its diagonal this ensures for free that $\mathbf{D}$ is indeed diagonal. Then, we run the CheckZero protocol (Algorithm 10) to show that the commitments obtained at the end of the protocol all open to 0, i.e. that Equation (11) is satisfied. The whole proof of knowledge is described in Algorithm 14 (prover's view) and Algorithm 15 (verifier's view). In Algorithm 14, the prover takes the following inputs:

- the secret monomial matrix $\mathbf{Q} \in \mathsf{M}_n(p)$ represented by n integers $\mathsf{pos}_0, ..., \mathsf{pos}_{n-1} \in [0, n-1]$, representing the position of the non-zero entries in each row of $\mathbf{Q}$, and n field elements $c_0, ..., c_{n-1} \in \mathbb{F}_p$, corresponding to the values of the non-zero entries of $\mathbf{Q}$. Both the positions and the constants are ordered with respect to the rows;
- the public generator matrix $\mathbf{G}_1 \in \mathbb{F}_p^{k\times n}$ and parity-check matrix $\mathbf{H}_2 \in \mathbb{F}_p^{(n-k)\times n}$;
- $n \cdot d$ random VOLE correlations $[\![u_{i,j}]\!]$, for random $u_{i,j} \in \mathbb{F}_p, v_{i,j} \in \mathbb{F}_{p^\tau}$, represented as polynomials $f_{u_{i,j}}(X) = u_{i,j}X + v_{i,j}$ for $(i,j) \in [0, n-1] \times [0, d-1]$;
- n random VOLE correlations $[\![s_i]\!]$, for random $s_i \in \mathbb{F}_p, \sigma_i \in \mathbb{F}_{p^\tau}$, represented as polynomials $f_{s_i}(X) = s_iX + \sigma_i$, $\forall i \in [0, n-1]$;
- n random VOLE correlations $[\![\overline{s}_i]\!]$, for random $\overline{s}_i \in \mathbb{F}_p, \overline{\sigma}_i \in \mathbb{F}_{p^\tau}$, represented as polynomials $f_{\overline{s}_i}(X) = \overline{s}_iX + \overline{\sigma}_i$, $\forall i \in [0, n-1]$;
- $2d$ random VOLE correlations $[\![a_\ell]\!]$, for random $a_\ell, b_\ell \in \mathbb{F}_{p^\tau}$, represented as polynomials $f_{a_\ell}(X) = a_\ell X + b_\ell$, $\forall \ell \in [0, 2d-1]$;
- a random seed $\mathtt{sd} \in \mathbb{F}_{2^\lambda}$.

Then, it outputs a proof that $\mathbf{Q}$ is a monomial matrix satisfying Equation (11). Instead, in Algorithm 15 the verifier takes as input:

- the public generator matrix $\mathbf{G}_1 \in \mathbb{F}_p^{k\times n}$ and parity-check matrix $\mathbf{H}_2 \in \mathbb{F}_p^{(n-k)\times n}$;
- masked secrets $\mathbf{t}$, $\mathbf{e}$, $\overline{\mathbf{e}}$;
- the point $\Delta \in \mathbb{F}_{p^\tau}$;
- the degree-$(2d+1)$ polynomial for the CheckZero verification, $[\![A]\!]^{(2d+1)}$;
- $r_{u_{i,j}} = f_{u_{i,j}}(\Delta)$, $r_{s_j} = f_{s_j}(\Delta)$, $r_{\overline{s}_j} = f_{\overline{s}_j}(\Delta)$, $r_{\overline{s}_j} = f_{\overline{s}_j}(\Delta)$, $r_{a_\ell} = f_{a_\ell}(\Delta)$;
- the seed $\mathtt{sd}$ for generating the coefficients of the linear combinations.

Remark 18 (Improve CheckZero verification step) Notice that the prover would need to employ the CheckZero verification step $2n+n^2$ times (for each $[\![\mathsf{ColCheck}_i]\!]^{(2d)}$, $[\![\mathsf{ConstCheck}_i]\!]^{(2)}$ and $[\![y_{i,j}]\!]^{(2d+1)}$). To avoid a major communication overhead, we consider a random linear

combination of all these polynomials and compute a single instance of CheckZero on the latter. However, we need all the polynomials to have the same degree $(2d+1)$: hence, we correct the polynomials having a smaller degree and then compute the linear combination (as shown in line 25 of Algorithm 14). Alternatively, we could split the verification steps based on the degree of the polynomials, take random linear combinations inside each group and compute many instances of the CheckZero protocol. However, this approach slightly increases the size of the proof.

Proposition 12 *The Model 3 correctly characterize a monomial* $\mathbf{Q}$ *matrix such that* $\mathbf{G}_2 = \mathbf{S}\mathbf{G}_1\mathbf{Q}$ *for some invertible matrix* $\mathbf{S}$ *and given generator matrixes* $\mathbf{G}_1, \mathbf{G}_2$.

Proof This is a direct consequence of Proposition 10 and 11. Indeed, they prove that $\mathbf{Q} = \mathbf{DP}$ for some diagonal matrix $\mathbf{D}$ and permuation matrix $\mathbf{P}$. The final check proves the relation $\mathbf{G}_1\mathbf{DPH}_2^\top$ holds, and thus we have the thesis. □

Algorithm 14 $\mathsf{P.CheckLEP}\big(\mathbf{Q}, \mathsf{pk} = (\mathbf{G}_1, \mathbf{H}_2), (\llbracket u_{i,j} \rrbracket)_{i\in[0,n-1],j\in[0,d-1]}, (\llbracket s_i \rrbracket)_{i\in[0,n-1]}, (\llbracket \overline{s}_i \rrbracket)_{i\in[0,n-1]}, (\llbracket a_\ell \rrbracket)_{\ell\in[0,2d-1]}\big)$

Compute VOLE correlation for the rows of $\mathbf{P}$ such that $\mathbf{Q} = \mathbf{DP}$

for $i \in [0, n-1]$ **do**

  $(\mathbf{t}_i, \llbracket \mathbf{z}_i \rrbracket^{(2d)}, \llbracket \mathsf{PosCheck_i} \rrbracket^{(2)}) \leftarrow \mathsf{P.VOLE-ElementaryVector}(\mathsf{pos}_i, (\llbracket u_{i,j} \rrbracket)_{j\in[0,d-1]})$

Compute $\mathbf{P}$'s columns check

for $j \in [0, n-1]$ **do**

  $\llbracket \mathsf{ColSum}_j \rrbracket^{(2d)} = \sum_{i=0}^{n-1} \llbracket z_{i,j} \rrbracket^{(2d)}$

  $\llbracket \mathsf{ColCheck}_j \rrbracket^{(2d)} = \llbracket \mathsf{ColSum}_j \rrbracket^{(2d)} - X^{2d}$

Commit to the constants $c_0, ..., c_{n-1}$ and compute their check.

for $i \in [0, n-1]$ **do**

  $e_i = c_i - s_i$

  $\llbracket c_i \rrbracket = f_{c_i}(X) = c_i X + r_i$

  $\overline{c}_i = c_i^{p-2}$

  $\overline{e}_i = \overline{c}_i - \overline{s}_i$

  $\llbracket \overline{c}_i \rrbracket = f_{\overline{c}_i}(X) = \overline{c}_i X + \overline{\sigma}_i$

  $\llbracket \mathsf{ConstCheck}_i \rrbracket^{(2)} = f_{c_i}(X) \cdot f_{\overline{c}_i}(X) - X^2$

Compute VOLE correlations for $\mathbf{W} = \mathbf{DPH}_2^\top$

for $j \in [0, n-k-1]$ **do**

  for $i \in [0, n-1]$ **do**

    $\llbracket w_{i,j} \rrbracket^{(2d+1)} = \llbracket c_i \rrbracket \cdot \sum_{m=0}^{n-1} \left(\llbracket \mathbf{z}_{i,m} \rrbracket^{(2d)} \cdot (\mathbf{H}_2)_{j,m} \right)$

Compute the VOLE correlations for $\mathbf{Y} = \mathbf{G}_1\mathbf{W}$

for $j \in [0, n-k-1]$ **do**

  for $i \in [0, k-1]$ **do**

    $\llbracket y_{i,j} \rrbracket^{(2d+1)} = \sum_{m=0}^{n-1} \left((\mathbf{G}_1)_{i,m} \cdot \llbracket \mathbf{w}_{m,j} \rrbracket^{(2d+1)} \right)$

  $\gamma^j \xleftarrow{\$,\mathtt{sd}} (\mathbb{F}_{p^\tau}^*)^k$

  $f_j(X) = \sum_{i=0}^{k-1} \left(\gamma_i^j \cdot \llbracket y_{i,j} \rrbracket^{(2d+1)} \right)$

Merge polynomials and perform CheckZero

$\mathbf{t} = (\mathbf{t}_0, .., \mathbf{t}_{n-1})$

$\mathbf{e} = (e_0, ..., e_{n-1})$

$\overline{\mathbf{e}} = (\overline{e}_0, ...\overline{e}_{n-1})$

$\boldsymbol{\alpha} \xleftarrow{\$,\mathtt{sd}} (\mathbb{F}_{p^\tau}^*)^{4n}$

$F(X) = \sum_{i=0}^{n-1} (\alpha_i \cdot \llbracket \mathsf{ConstCheck}_i \rrbracket^{(2)} \cdot X^{2d-1}) + \sum_{i=0}^{n-1} (\alpha_{n+i} \cdot \llbracket \mathsf{PosCheck}_i \rrbracket^{(2)} \cdot X^{2d-1}) + \sum_{i=0}^{n-1} (\alpha_{n+i} \cdot \llbracket \mathsf{ColCheck}_i \rrbracket^{(2d)} \cdot X) + \sum_{i=0}^{n-1} (\alpha_{3n+i} \cdot f_i(X))$

$\llbracket A \rrbracket^{(2d+1)} \leftarrow \mathsf{P.CheckZero}(F(X), (\llbracket a_\ell \rrbracket)_{\ell\in[0,2d-1]})$

return $\Pi = (\llbracket A \rrbracket^{(2d+1)}, \mathbf{t}, \mathbf{e}, \overline{\mathbf{e}}, \mathtt{sd})$

Algorithm 15 $\mathsf{V.CheckLEP}\big(\Pi, \mathsf{pk} = (\mathbf{G}_1, \mathbf{H}_2), (r_{u_{i,j}})_{i\in[0,n-1],j\in[0,d-1]}, (r_{s_j})_{j\in[0,n-1]}, (r_{\overline{s}_j})_{j\in[0,n-1]}, (r_{a_\ell})_{\ell\in[0,d-1]}, \Delta\big)$

Compute VOLE correlation for the rows of $\mathbf{P}$ such that $\mathbf{Q} = \mathbf{DP}$

for $i \in [0, n-1]$ **do**

  $(r_{\mathbf{z}_i}, r_{\mathsf{PosCheck}_i}) \leftarrow \mathsf{V.VOLE-ElementaryVector}(\mathbf{t}_i, (r_{u_{i,j}})_{j\in[0,d-1]}, \Delta)$

Compute $\mathbf{P}$'s columns check

for $j \in [0, n-1]$ **do**

  $r_{\mathsf{ColSum}_j} = \sum_{i=0}^{n-1} r_{z_{i,j}}$

  $r_{\mathsf{ColCheck}_j} = r_{\mathsf{ColSum}_j} - \Delta^{2d}$

Compute the VOLE correlations for the constants and for the constants' check.

for $i \in [0, n-1]$ **do**

  $r_{c_i} = e_i \cdot \Delta + r_{s_i}$

  $r_{\overline{c}_i} = \overline{e}_i \cdot \Delta + r_{\overline{s}_i}$

  $r_{\mathsf{ConstCheck}_i} = r_{c_i} \cdot r_{\overline{c}_i} - \Delta^2$

Compute the VOLE correlations for $\mathbf{W} = \mathbf{DPH}_2^\top$.

for $j \in [0, n-k-1]$ **do**

  for $i \in [0, n-1]$ **do**

    $r_{w_{i,j}} = r_{c_i} \cdot \sum_{m=0}^{n-1} \big(r_{z_{i,m}} \cdot (\mathbf{H}_2)_{j,m}\big)$

Compute the VOLE correlations for $\mathbf{Y} = \mathbf{G}_1\mathbf{W}$.

for $j \in [0, n-k-1]$ **do**

  for $i \in [0, k-1]$ **do**

    $r_{y_{i,j}} = \sum_{m=0}^{n-1} \big((\mathbf{G}_1)_{i,m} \cdot r_{w_{m,j}}\big)$

  $\gamma^j \xleftarrow{\$,\mathtt{sd}} (\mathbb{F}_{p^\tau}^*)^k$

  $r_j = \sum_{i=0}^{k-1} (\gamma_i^j \cdot r_{y_{i,j}})$

Merge the results and compute the output

$\boldsymbol{\alpha} \xleftarrow{\$,\mathtt{sd}} (\mathbb{F}_{p^\tau}^*)^{4n}$

$r = \sum_{i=0}^{n-1} (\alpha_i \cdot r_{\mathsf{ConstCheck}_i} \cdot \Delta^{2d-1}) + \sum_{i=0}^{n-1} (\alpha_{n+i} \cdot r_{\mathsf{PosCheck}_i} \cdot \Delta^{2d-1}) + \sum_{i=0}^{n-1} (\alpha_{2n+i} \cdot r_{\mathsf{ColCheck}_i} \cdot \Delta) + \sum_{i=0}^{n-1} (\alpha_{3n+i} \cdot r_j)$

$B \leftarrow \mathsf{V.CheckZero}(\llbracket A \rrbracket^{(2d+1)}, r, (r_{a_\ell})_{\ell\in[0,2d-1]})$

References

[1] Alagic, G., Bros, M., Ciadoux, P., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.-K., Miller, C., et al.: Status report on the first round of the additional digital signature schemes for the nist post-quantum cryptography standardization process. NIST IR **8528** (2024)

[2] Baum, C., Beullens, W., Braun, L., Saint Guilhem, C.D., Klooß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: FAEST v2: Algorithm Specifications (Version 2.0) (2024). https://faest.info/faest-spec-v2.0.pdf

[3] Baum, C., Braun, L., Saint Guilhem, C., Klooß, M., Orsini, E., Roy, L., Scholl, P.: Publicly verifiable zero-knowledge and post-quantum signatures from vole-in-the-head. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, pp. 581–615. Springer, Cham (2023)

[4] Bettaieb, S., Bidoux, L., Gaborit, P., Kulkarni, M.: Modelings for generic PoK and Applications: Shorter SD and PKP based Signatures. Cryptology ePrint Archive, Paper 2024/1668 (2024). https://eprint.iacr.org/2024/1668

[5] Adj, G., Aragon, N., Barbero, S., Bardet, M., Bellini, E., Bidoux, L., Chi-Domínguez, J., Dyseryn, V., Esser, A., Feneuil, T., Gaborit, P., Neveu, R., Rivain, M., Rivera-Zamarripa, L., Sanna, C., Tillich, J., Verbel, J., Zweydinger, F.: Mirath (merger of MIRA/MiRitH). Technical report, National Institute of Standards and Technology (2024). available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures

[6] Aragon, N., Bardet, M., Bidoux, L., Chi-Domínguez, J., Dyseryn, V., Feneuil, T., Gaborit, P., Joux, A., Neveu, R., Rivain, M., Tillich, J., Vinçotte, A.: RYDE. Technical report, National Institute of Standards and Technology (2024). available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures

[7] Aguilar Melchor, C., Bettaieb, S., Bidoux, L., Feneuil, T., Gaborit, P., Gama, N., Gueron, S., Howe, J., Hülsing, A., Joseph, D., Joux, A., Kulkarni, M., Persichetti, E., Randrianarisoa, T.H., Rivain, M., Yue, D.: SDitH — Syndrome Decoding in the Head. Technical report, National Institute of Standards and Technology (2024). available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures

[8] Barenghi, A., Biasse, J.-F., Persichetti, E., Santini, P.: 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, pp. 23–43 (2021). Springer

[9] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86, pp. 186–194. Springer, Berlin, Heidelberg (1987)

[10] Saeed, M.A.: Algebraic Approach for Code Equivalence. Theses, Normandie Université ; University of Khartoum (December 2017). https://theses.hal.science/tel-01678829

[11] Gilchrist, V., Marco, L., Petit, C., Tang, G.: On the security of two blind signatures from code equivalence problems. Cryptology ePrint Archive, Paper 2025/1883 (2025). https://eprint.iacr.org/2025/1883

[12] Abdalla, M., An, J.H., Bellare, M., Namprempre, C.: From identification to signatures via the fiat-shamir transform: Minimizing assumptions for security and forward-security. In: Knudsen, L.R. (ed.) Advances in Cryptology — EUROCRYPT 2002, pp. 418–433. Springer, Berlin, Heidelberg (2002)

[13] Battagliola, M., Longo, R., Pintore, F., Signorini, E., Tognolini, G.: A Revision of CROSS Security: Proofs and Attacks for Multi-Round Fiat-Shamir Signatures. Cryptology ePrint Archive, Paper 2025/127 (2025). https://eprint.iacr.org/2025/127

[14] Barenghi, A., Biasse, J.-F., Persichetti, E., Santini, P.: On the computational hardness of the code equivalence problem in cryptography. Advances in Mathematics of Communications **17**(1), 23–55 (2023)

[15] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing Vector OLE. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS '18, pp. 896–912. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243868 . https://doi.org/10.1145/3243734.3243868

[16] Roy, L.: SoftSpokenOT: Communication–Computation Tradeoffs in OT Extension. Cryptology ePrint Archive, Paper 2022/192 (2022). https://eprint.iacr.org/2022/192

[17] Baum, C., Braun, L., Saint Guilhem, C.D., Klooß, M., Majenz, C., Mukherjee, S., Ramacher, S., Rechberger, C., Orsini, E., Roy, L., Scholl, P.: FAEST: Algorithm Specifications (Version 1.1) (2023). https://faest.info/faest-spec-v1.1.pdf

[18] Baum, C., Beullens, W., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: One tree to rule them all: Optimizing ggm trees and owfs for post-quantum signatures. In: Chung, K.-M., Sasaki, Y. (eds.) Advances in Cryptology – ASIACRYPT 2024, pp. 463–493. Springer, Singapore (2025)

[19] Yang, K., Sarkar, P., Weng, C., Wang, X.: Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In:

Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. CCS '21, pp. 2986–3001. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484556 . https://doi.org/10.1145/3460120.3484556

[20] Caminata, A., Cartor, R., Meneghetti, A., Mora, R., Pellegrini, A.: Quadratic modelings of syndrome decoding. In: Niederhagen, R., Saarinen, M.-J.O. (eds.) Post-Quantum Cryptography, pp. 35–70. Springer, Cham (2025)