SETH: Subcode Equivalence in the Head

Marco Baldi[1†], Michele Battagliola[1,2†], Rahmi El Mechri[1,3†], Laura Mattiuz[1*†], Marco Rinaudo[4†], Paolo Santini[1†], Riccardo Schiavoni[1†], Edoardo Signorini[5†]

[1*]Department of Information Engineering, Marche Polytechnic University, Ancona, Italy.
[2]School of Computer Science, St. Gallen University, St. Gallen, Switzerland.
[3]IMT School for Advanced Studies, Lucca, Italy.
[4]Turin Polytechnic University, Turin, Italy.
[5]European Commission, Joint Research Centre (JRC), Ispra, Italy.

*Corresponding author(s). E-mail(s): l.mattiuz@pm.univpm.it;
Contributing authors: m.baldi@staff.univpm.it;
battagliola.michele@proton.me; rahmi.elmechri@imtlucca.it;
marco.rinaudo@polito.it; p.santini@staff.univpm.it;
r.schiavoni@pm.univpm.it; edoardo.signorini@ec.europa.eu;
†These authors contributed equally to this work.

Abstract

The Multi-Party Computation in the Head (MPCitH) paradigm has proven to be a versatile tool to design post-quantum digital signature schemes. Recently, MPCitH was improved with Threshold Computation in the Head (TCitH) and Vector Oblivious Linear Evaluation in the Head (VOLEitH). In this paper, we propose two polynomial models for the Subcode Equivalence Problem (SEP) and derive TCitH and VOLEitH signatures from them. The resulting scheme has better performance when compared to other SEP-based signature schemes and also features shorter signatures when compared to other schemes based on similar assumptions.

Keywords: Digital Signatures, Code-based cryptography, Subcode Equivalence, VOLEitH, TCitH, Zero-Knowledge Proofs

1 Introduction

Following the standardization of the post-quantum digital signature schemes CRYSTALS-Dilithium, FALCON, and SPHINCS$^+$, NIST issued a call for additional digital signature schemes in September 2022, with the goal of diversifying the ecosystem of post-quantum signature standards [1]. Among the over 40 initial submissions, in May 2026, NIST announced that 9 submissions are going to proceed to the third round of evaluation, that is currently ongoing. Among these, three are based on the Multi-party Computation in-the-Head (MPCitH) or the Vector Oblivious Linear Evaluation in-the-Head (VOLEitH) frameworks [2–4], whereas only one is a code-based scheme [4] (while also employing the VOLEitH technique).

MPCitH and TCitH

Originally proposed in [5] by Ishai et al., the MPCitH approach is a general way for building Honest Verifier Zero Knowledge (HVZK) arguments on top of Multi-Party Computation (MPC) protocols. In a nutshell, to prove the knowledge of a *witness* w such that $f(w) = x$ for some public x and f, the prover shares w "in its head" and performs an MPC protocol computing $f(w)$ on these shares. For each party, the prover commits to the party's view and sends the commitment to the verifier, alongside all broadcasts made in the MPC protocol and the output shares. Then, the verifier asks for a party i and the prover opens all the shares of all parties except i. In this way, the verifier can check that the shares are consistent with the committed values and that the MPC protocol was done correctly. TCitH [6] is an improvement of MPCitH, where the prover uses some threshold secret sharing protocol (such as Shamir's secret sharing [7]) instead of additive secret sharing. This allows opening fewer parties and also reducing the soundness error, thus achieving shorter signatures.

VOLEitH

VOLE-based proofs use random VOLE correlations in a commit-and-prove framework where the VOLE correlations are used to commit to the witness, and then the statement is proved exploiting the homomorphic properties of the commitment. 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 [8]: it employs an all-but-one vector commitment scheme to generate VOLE correlations, and allows the construction of publicly verifiable ZK proofs while keeping the structure of VOLE-based protocols. The first example of VOLEitH signature is FAEST [3], and this approach was later adopted by both SDitH [4] and PERK [9], another NIST submission.

Subcode Equivalence Problem

The Subcode Equivalence Problem (SEP) is very similar to the more common Permutation Equivalence Problem. In a nutshell, for two given linear block codes, instead of asking to retrieve a permutation mapping one of the two codes into the other, SEP requires to find a permutation sending one of the two codes into a subcode of the other.

Despite being recently shown to be NP-complete [10], SEP has seen limited application in cryptography: as of today, the only protocol based on it is the identification scheme presented in [10], which is a variant of the Girault one [11] and that suffers the same efficiency problems, since it requires the prover to send multiple matrices. This is also the case for SEP in the rank metric: indeed, the Matrix Subcode Equivalence Problem (MSEP) was also shown to be NP-complete and has been used only once to construct a VOLEitH signature scheme [12].

A variant of SEP is the Permuted Kernel Problem (PKP), where the dimension of the subcode is 1 (i.e. the subcode is generated by a single vector). PKP or some of its variants are at the basis of several popular digital signature algorithms, including the NIST submission PERK [9] and the recently proposed VOLEitH-based signature scheme in [13].

Our Contribution

Following a similar approach to [13] and [14], we propose two polynomial modellings for SEP. Then, using the Polynomial Interactive Oracle Proof formalism, we are able to derive TCitH and VOLEitH signatures from them. Our signatures have better performance when compared to other SEP-based signatures like the ones presented in [10, 12] and achieve similar performance when compared to other modern signatures from similar assumptions, like Linear Code Equivalence or PKP [9, 13, 15]. Compared to the recent work [14], that tackles the Linear Code Equivalence Problem (LEP) with a similar approach, our signature offers the following advantages:

- while the exact security of LEP is still uncertain [16, 17], SEP is NP-hard, offering higher security guarantees;
- SEP allows for better parameters, in particular because it employs subcodes, allowing for shorter keys and also a smaller field-size;
- using permutations allows for an easier modelling, which requires fewer VOLE correlations and thus leads to shorter and faster signatures compared to [14].

1.1 Paper Organization

In Section 2 we introduce the notation used in this work and the preliminaries. In Section 3 we describe the two polynomial modellings for the Subcode Equivalence Problem that we will use to design our signatures. In Section 4 we present our signature scheme, its parameters and provide details about the performance. Finally, in Section 5 we draw some conclusions and pinpoint possible areas for improvements and future works. As to the appendices, Appendices A and B contain the details about the generation and commitment procedures to degree-1 polynomials using GGM trees, Appendix C describes more thoroughly the consistency check for VOLEitH and, lastly, Appendix D contains additional functions for type conversion.

2 Preliminaries

In this section, we establish the notation we use throughout the paper and recall the necessary background concepts about linear codes, digital signatures and the VOLEitH and TCitH frameworks.

2.1 Notation

Given two integers $a, b \in \mathbb{Z},\ a \leq b$, we write $[a, b] = \{a, a+1, \ldots, b\}$; if $a = 1$, we simply write $[b] = [1, b]$.

In this paper, q denotes a prime power and $\mathbb{F}_q$ indicates the finite field with q elements. We will use boldface uppercase letters to denote matrices, and boldface lowercase letters to denote vectors. Given a matrix $\mathbf{M}$, we denote by $\mathbf{M}_{i,:}$ its i-th row and by $\mathbf{M}_{:,i}$ its i-th column. The identity matrix of size k is indicated as $\mathbf{I}_k$. We use the following groups of matrices over $\mathbb{F}_q$:

- $\mathsf{GL}_k(\mathbb{F}_q)$ stands for the group of invertible $k \times k$ matrices over $\mathbb{F}_q$;
- S_n is the symmetric group on n elements, i.e., the set of permutations of length n. With abuse of notation, we will denote an element in S_n both as an $n \times n$ matrix $\mathbf{P}$ and as a function π.

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 seed, we will write $s \xleftarrow{\$,\mathsf{seed}} S$.

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 by R and hence the output is uniquely determined.

2.2 Subcode Equivalence Problem

A $[n, k]_q$ *linear code* $\mathscr{C}$ is a k-dimensional subspace of $\mathbb{F}_q^n$: n is the length while k is the dimension of the code. An element $\mathbf{c} \in \mathscr{C}$ is called codeword. We define *code rate* the quantity $R = k/n$. A code $\mathscr{C} \subseteq \mathbb{F}_q^n$ 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 $\mathscr{C}^\perp$. A basis for $\mathscr{C}^\perp$ 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 $\mathbf{G}\mathbf{H}^\top = \mathbf{0}$.

For any $\mathbf{S} \in \mathsf{GL}_k(\mathbb{F}_q)$, $\mathbf{G}$ and $\mathbf{SG}$ generate the same code; the same applies to parity-check matrices, with changes of basis taken from $\mathsf{GL}_{n-k}(\mathbb{F}_q)$.

A *systematic* generator matrix is in the form $(\mathbf{I}_k, \mathbf{A})$ with $\mathbf{A} \in \mathbb{F}_q^{k \times (n-k)}$, from which we obtain the systematic parity-check matrix as $(\mathbf{B}, \mathbf{I}_{n-k})$, with $\mathbf{B} = -\mathbf{A}^\top$. We indicate as SF the function that, on input a generator matrix, returns the systematic form.

For some codes, the first k columns of a generator matrix may form a singular matrix, hence the systematic form may not exist. In such cases, the notion of systematic form may be generalised through that of "standard" form, which allows for

the $k \times k$ identity matrix to show up somewhere among the columns of $\mathbf{G}$; the same applies, with some evident adjustments, to parity-check matrices. For the sake of simplicity, in this paper we implicitly assume that all the matrices we consider admit a systematic form.

The main goal of the paper is to design a digital signature based on the Subcode Equivalence Problem (SEP), which reads as follows:

Problem 1 (Subcode Equivalence Problem)**.** Given two codes $\mathscr{C}, \mathscr{C}' \subseteq \mathbb{F}_q^n$ with respective dimension k and $k' < k$, find, if it exists, a permutation $\pi \in \mathsf{S}_n$ such that $\pi(\mathscr{C}') \subset \mathscr{C}$ (i.e. $\pi(\mathscr{C}')$ is a subcode of $\mathscr{C}$).

The above problem can be rephrased in terms of generator and parity-check matrices by asking, given a generator matrix $\mathbf{G}' \in \mathbb{F}_q^{k' \times n}$ and a parity check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, to find a permutation $\mathbf{P} \in \mathsf{S}_n$ such that $\mathbf{G}'\mathbf{P}\mathbf{H}^\top = \mathbf{0}$.

Hardness of SEP

We recall that SEP is NP-complete [10]. Recalling [18], the hardest instances for SEP are obtained when both $\mathscr{C}$ and $\mathscr{C}'$ do not have repeated entries, and when there is on average a unique solution. For random codes, this happens when[1]

$$\frac{(n! - 1) \cdot \begin{bmatrix} k \\ k' \end{bmatrix}_q}{\begin{bmatrix} n \\ k' \end{bmatrix}_q} < 1,$$

where

$$\begin{bmatrix} k \\ k' \end{bmatrix}_q = \frac{(1-q^k)(1-q^{k-1})\cdots(1-q^{k-k'+1})}{(1-q)(1-q^2)\cdots(1-q^{k'})}$$

is the Gaussian binomial coefficient. Indeed, given a subcode $\mathscr{C}'$, there are at most $n!$ distinct codes we can obtain by permuting the coordinates of $\mathscr{C}'$. For each of these subspaces, we estimate as $\begin{bmatrix} k \\ k' \end{bmatrix}_q / \begin{bmatrix} n \\ k' \end{bmatrix}_q$ the probability that it is indeed a subcode of $\mathscr{C}$. Indeed, the numerator counts the number of subcodes in $\mathscr{C}$, while the denominator counts the overall number of k'-dimensional subcodes of $\mathbb{F}_q^n$.

Asymptotically, for $k = Rn$ and constant k', the above inequality turns into

$$q = \left(\frac{n}{e}\right)^{\frac{1}{k'(1-R)}} \cdot \big(1 + o(1)\big), \tag{1}$$

where $R = \frac{k}{n}$ is the code rate.

The basic attack on SEP is the so-called KMP algorithm [20] which, essentially, corresponds to a brute force search over all permutations, sped up by a meet-in-the-middle approach and by exploiting the linear constraints imposed by the parity-check

[1] We here recall the heuristic from [18], even though a more precise analysis is given in [19]. The difference between the two approaches is rather mild and even vanishes asymptotically.

equations. When q is chosen as previously shown, this approach has, asymptotically, cost given by $2^{c \cdot n \cdot \left(1+o(1)\right)}$, where

$$c = \frac{1}{1+R} \cdot \left(\log_2(1+R) + R \cdot \log_2 \left(\frac{n}{e} \right) \right). \tag{2}$$

This algorithm can be slightly improved with the approaches in [18] and [21], for which, however, closed-form formulas are not provided. Without entering into further details, the high level idea consists in applying KMP to a more convenient parity-check matrix, obtained with row operations from $\mathbf{H}$. Here, more convenient means that a subset of the parity-check equations involve a (small) subset of coordinates. In Section 4.4 we use both [18, 21] to provide the parameters for our protocol.

2.3 Interactive Proofs and Zero-Knowledge Protocols

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

In the following we are only interested in *hard relations*, i.e. binary relations such that, given x, it is infeasible to efficiently find a corresponding witness w. In particular we are interested in the relation induced by the Subcode Equivalence Problem: X is the set of pairs $(\mathscr{C}, \mathscr{C}')$ where $\mathscr{C}$ is a $[n,k]_q$ code and $\mathscr{C}'$ is a $[n,k']_q$ code with $k' < k$, $W = \mathsf{S}_n$ and

$$L_{\mathsf{SEP}} = \{(\mathscr{C}, \mathscr{C}') \in X \mid \exists \pi \in \mathsf{S}_n \text{ s.t. } \pi(\mathscr{C}') \subset \mathscr{C}\}.$$

Definition 2 (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 $(x, w) \in R$, 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 $(\mathcal{P}(w), \mathcal{V})(x)$.

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).

We are only interested in protocols where the prover $\mathcal{P}$ sends the first and last message within an execution of an interactive proof. We refer to an interactive proof having $2\mu + 1$ rounds of communication as a $2\mu + 1$-*round interactive proof*.

Definition 3 (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 its random choices. In this context, the messages sent by the verifier are called *challenges*, denoted ch_i. In each round of communication, the verifier samples the challenge ch_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 $(\mathcal{P}, \mathcal{V})$

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

In the following sections, we will often refer to a property of *soundness* of the interactive protocol underlying our signature scheme. Intuitively, the soundness property ensures that, with high probability, a prover cannot convince the verifier of the validity of an invalid statement.

Definition 4 (Soundness)**.** An interactive proof $(\mathcal{P}, \mathcal{V})$ for a binary relation R is said to be *sound* if the probability of an invalid input being accepted is negligible. Formally, the protocol is sound if for every $x \notin L_R$ and for every prover $\mathcal{P}^*$, we have:

$$\mathsf{Pr}[(\mathcal{P}^*, \mathcal{V})(x) = 1] \leq \epsilon(x)$$

where the value $\epsilon(x)$ is negligible in $|x|$. The value $\epsilon(x)$ is called *soundness error*.

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;
- $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp}, \lambda)$: is a probabilistic *key generation* algorithm that, given as inputs the public parameters pp and the security parameter λ, outputs a pair of public and private keys $(\mathsf{pk}, \mathsf{sk})$;

– $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, \mathsf{m})$: is a (usually) probabilistic *signature* algorithm that takes as input a private key sk and a message m, and outputs a signature σ for m;
– $b \leftarrow \mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \sigma)$: is a deterministic algorithm that takes as inputs a public key pk, a message m and a signature σ, and outputs a bit $b \in \{0, 1\}$, where $b = 1$ means "accept" (i.e. that σ is a valid signature for m and pk) and $b = 0$ means "reject".

Moreover, it is required that $\Pr[\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \sigma) = 1 \mid \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{m})] = 1$.

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 $\mathsf{O}^{DS}(\cdot)$. Define the following experiment where M denotes the set of the messages queried by $\mathcal{A}$ to O^{DS}.

$\mathsf{Exp}^{\text{EUF-CMA}}_{\mathsf{DS},\mathcal{A}}(\lambda)$:

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

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

Fiat-Shamir Transformation

Introduced in [22], 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 $(2\mu + 1)$-round interactive proof $(\mathcal{P}, \mathcal{V})$ for a hard relation $R \subseteq X \times W$, the resulting Fiat-Shamir signature is defined as in Algorithm 2. The work [23], later extended by [24] to cover the general multi-round case, shows the minimal conditions that an interactive proof should satisfy so that the obtained signature is secure.

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

$\mathsf{KeyGen}(\lambda)$

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

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

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

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

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

2.5 Polynomial Interactive Oracle Proofs

Polynomial Interactive Oracle Proofs (PIOP) [25] are interactive proofs where, in each round of interaction, the prover provides the verifier with oracle access to a multivariate polynomial function of bounded degree. Then, the verifier may ask the evaluation of the polynomial on arbitrary points of its choice. While originally developed for designing SNARKs, the PIOP formalism allows unifying the TCitH framework (which was originally introduced using secret sharing) with the VOLEitH one as follows.

Let $(\mathbf{x}, \mathbf{w})$ be an instance of a hard relation R, with ℓ being the length of $\mathbf{w} = (w_1, \ldots, w_\ell)^\top \in \mathbb{F}_q^\ell$. Moreover, let $F = \{f_1, ..., f_t\}$ be a system of polynomial equations where $f_i \in \mathbb{F}_{q^r}[X_1, ..., X_\ell]$ has degree at most D and such that $F(\bar{\mathbf{w}}) = 0$ if and only if $(\mathbf{x}, \bar{\mathbf{w}}) \in R$. Finally, let us assume that the prover has a way to commit to polynomials and to provably open some evaluations later, while keeping hidden the other evaluations. From a high-level perspective, a PIOP is structured as follows:

1. The prover generates ℓ degree-1 polynomials $p_j(X) = w_j X + v_j,\ j = 1, \ldots, \ell$ for some random $v_1, ..., v_\ell \in \mathbb{F}_{q^r}$. Moreover the prover generates additional $D-1$ degree-1 polynomials $\bar{p}_i(X) = \bar{u}_i X + \bar{v}_i, \bar{u}_i, \bar{v}_i \in \mathbb{F}_{q^r}, i = 1, ..., D-1$.[2] Finally, the prover commits to those polynomials and sends the commitment com to the verifier.
2. The verifier picks a random t-tuple $(\gamma_1, ..., \gamma_t)$ of elements in $\mathbb{F}_{q^r}$ and sends them to the prover.
3. The prover computes the polynomial $\bar{f}_i = X^D f_i(\frac{p_1(X)}{X}, ..., \frac{p_\ell(X)}{X})$ and computes $\bar{F} = \sum \gamma_i \bar{f}_i$. Notice that, by construction, $\deg \bar{F} \leq D-1$ since, for all $i \in [t]$, $\bar{f}_i(X) = f_i(\mathbf{w}) \cdot X^D + \ldots$, and $F(\mathbf{w}) = 0$. The prover finally computes the polynomials $mask(X) = \sum_{i=1}^{D-1} (\bar{u}_i X + \bar{v}_i) X^{i-1}$ and $F_{\mathsf{masked}}(X) = \bar{F}(X) + mask(X)$ and sends it to the verifier.
4. The verifier picks a random $\Delta \in \mathbb{F}_{q^r}$ and sends it to the prover.

[2] In the VOLE formalism these degree-1 polynomials are called VOLE correlations.

5. The prover sends to the verifier $d_j = p_j(\Delta)$ for all $j \in [\ell]$, $\bar{d}_i = \bar{p}_i(\Delta)$ for all $i \in [D-1]$ and a proof π proving that the evaluations are consistent with the initial commitment com.
6. The verifier checks π and that $\deg(F_{\mathsf{masked}}) \leq D-1$. Then computes $\bar{F}(\Delta)$ using $\{d_j\}_{j\in[\ell]}$ and $mask(\Delta) = \sum_{i=1}^{D-1} \bar{d}_i \Delta^{i-1}$ and checks that $\bar{F}(\Delta) + mask(\Delta) = F_{\mathsf{masked}}(\Delta)$.

It is possible to prove that the above protocol has soundness error of

$$\frac{1}{q^r} + \left(1 - \frac{1}{q^r}\right) \cdot \frac{D}{q^r}.$$

This is due to the fact that a malicious prover can succeed in two cases. If the prover is malicious, then there exists $\bar{i} \in [t]$ such that $f_{\bar{i}}(\mathbf{w}) \neq 0$, which implies that $\deg(\bar{f}_{\bar{i}}) = D$. Then, the polynomial F_{masked} passes the check on the degree with probability $1/q^r$, i.e. when $\gamma_{\bar{i}} = 0$. Alternatively, the malicious prover can still succeed if the second challenge Δ is a root of the polynomial $(\bar{F}(X) + mask(X)) - F_{\mathsf{masked}}(X)$, which has degree D. When $\mathbb{F}_{q^r}$ is a small field this would lead to a high soundness error and thus to a non-secure digital signature. In the context of PIOPs, TCitH and VOLEitH can be seen as techniques to reduce the soundness error of a PIOP alternative to (only) parallel repetitions.

Remark 1. Notice that we can consider $f_1, \ldots, f_t \in \mathbb{F}_{q^m}[X_1, \ldots, X_\ell]$ with $m \leq r$ and then lift them to an extension $\mathbb{F}_{q^r}$, in the case of TCitH, or $\mathbb{F}_{q^{r\tau}}$, in the case of VOLEitH: to achieve this, it must hold $m \mid r$ or $m \mid r\tau$, respectively. Then, the protocol proceeds as described above, by defining the polynomials $p_j(X)$ and $\bar{p}_i(X)$ over the extension field.

Threshold Computation-in-the-Head

In TCitH, the verifier samples as first challenge a random matrix $\Gamma \in \mathbb{F}_{q^r}^{\rho \times t}$ (corresponding to ρ t-tuples $(\gamma_1, \ldots, \gamma_t) \in \mathbb{F}_{q^r}^t$), and the prover computes ρ polynomials

$$F_{\mathsf{masked},i} = \sum_{j=1}^{t} \Gamma_{i,j} \bar{f}_j + \mathsf{mask}_i, \ i \in [\rho].$$

Notice that, to do so, the prover now has to compute $\rho \cdot (D-1)$ additional degree-1 polynomials $\{\bar{p}_{i,j}\}_{i\in[\rho],j\in[D-1]}$ to compute the ρ polynomials $\{mask_i\}_{i\in[\rho]}$. Instead, the polynomials $\bar{f}_1, \ldots, \bar{f}_t$ remain fixed. Then, the verifier samples as second challenge a single $\Delta \in \mathbb{F}_{q^r}$, which is used to compute the proof π and to perform all the checks. Hence, in this case the soundness error is

$$\frac{1}{q^{\rho r}} + \left(1 - \frac{1}{q^{\rho r}}\right) \cdot \frac{D}{q^r}$$

which is approximately D/q^r if $\rho r \log_2(q) \geq \lambda$. The soundness error is then further reduced by performing τ parallel repetitions of the whole protocol to achieve $(D/q^r)^\tau < 2^{-\lambda}$.

VOLE-in-the-Head

While TCitH still relies on parallel repetitions, in the VOLEitH framework the verifier samples the challenges $\gamma_1, \ldots, \gamma_t, \Delta$ from a set having an exponential number of elements. In particular, this requires to lift the polynomials $\{p_j\}_{j\in[\ell]}, \{\bar{p}_i\}_{i\in[D-1]}$ to an extension $\mathbb{F}_{q^{r\tau}}$ of $\mathbb{F}_{q^r}$ so that $q^{r\tau}$ is exponentially large (the technique is explained in the later pages). In this case, the soundness error is reduced to

$$\frac{1}{q^{r\tau}} + \left(1 - \frac{1}{q^{r\tau}}\right) \cdot \frac{D}{q^{r\tau}}. \tag{3}$$

Hence, parallel repetitions are not needed whenever τ is chosen such that $\frac{D}{q^{r\tau}} \approx 2^{-\lambda}$. This approach results in signatures having 7 rounds, since the lifting requires an additional round of communication. However, VOLEitH signatures are in general shorter than TCitH ones, since they do not incur the security loss due to parallel repetition [26]. In both cases, the signature is obtained applying the Fiat-Shamir transform to the interactive protocol. It remains to show how to generate the polynomials, how to commit to them and how to prove the consistency of the evaluations.

2.5.1 Generating Polynomials via GGM Trees

The initial polynomials are generated by the prover using GGM trees [27]. The GGM tree allows the prover to commit to $N = q^r$ seeds arranged as leaves of a binary tree by computing them by means of a length-doubling PRG, committing to each seed and then computing a digest of these commitments, which is sent to the verifier.Later, the prover can open all except one seed by sending only siblings of the nodes on the path from the root to the hidden leaf and the commitment to the hidden seed. This allows the verifier to recompute the digest and check if it matches, thus verifying the correctness of the $N-1$ opened seeds. We refer to Appendix A.1 for the detailed description of the commitment scheme obtained from GGM trees.

The polynomials and the relative evaluations can be obtained from the seeds as follows. Let PRG be a PRG having image $\mathbb{F}_q^\ell$, we can consider $\mathbf{s}^{(i)} = \mathsf{PRG}(\mathsf{seed}_i)$ for $i \in [0, N-1]$. The prover computes $\mathbf{u}, \mathbf{v}$ as

$$\mathbf{u} = \sum_{i=0}^{N-1} \mathbf{s}^{(i)} \in \mathbb{F}_q^\ell, \quad \mathbf{v} = -\sum_{i=0}^{N-1} i \cdot \mathbf{s}^{(i)} \in \mathbb{F}_{q^r}^\ell$$

Then, the prover computes the auxiliary value $\mathbf{t} = \mathbf{u} - \mathbf{w} = (t_1, \ldots, t_\ell)^\top$ and sets the polynomial $p_j(X) = (u_j - t_j)X + v_j = w_j X + v_j$. The commitment com is the hash of the seeds as well as $\mathbf{t}$.

The verifier chooses $\Delta \in [0, N-1]$ uniformly at random and receives all the seeds seed_i for $i \in [0, N-1] \setminus \{\Delta\}$ from the prover. Then, they can compute $p_j(\Delta)$ as:

$$p_j(\Delta) = -t_j\Delta + \sum_{i=0}^{N-1}(\Delta - i)\cdot s_j^{(i)} = -t_j\Delta + \Delta\sum_{i=0}^{N-1} s_j^{(i)} - \sum_{i=0}^{N-1} i\cdot s_j^{(i)} = (u_j - t_j)\Delta + v_j$$

When speaking about the VOLEitH technique, we refer to the tuple $((\mathbf{u}, \mathbf{v}), (\Delta, \mathbf{d}))$, with $\mathbf{d} = (p_1(\Delta), \ldots, p_\ell(\Delta))^\top$, as a *VOLE correlation*.

Remark 2 (Length of PRG output)**.** The procedure we have just described generates degree-1 polynomials whose leading coefficient belongs to $\mathbb{F}_q$ and whose constant term belongs to $\mathbb{F}_{q^r}$. However, the polynomials $(\bar{p}_i)_{i\in[D-1]}$ (used to compute the masking polynomial) must have both coefficients in $\mathbb{F}_{q^r}$. To achieve this, we employ a PRG having as image $\mathbb{F}_q^{\ell+(D-1)\cdot r}$. Then, the first ℓ elements of each PRG-output are used as above to compute the polynomials $(p_j)_{j\in[\ell]}$. Instead, the remaining $(D-1)\cdot r$ elements are divided into $(D-1)$ r-tuples, and combined into elements of $\mathbb{F}_{q^r}$. Taking into account this observation, TCitH and VOLEitH require two different PRG outputs:

- since TCitH involves ρ masking polynomials $mask_i$ in each round of the protocol, the length of the PRG output must be set to $\ell_{\mathsf{TC}} = \ell + \rho \cdot r \cdot (D-1)$;
- since VOLEitH works with polynomials in a larger field $\mathbb{F}_{q^{r\tau}}$, the length of the PRG output for VOLEitH must be set to $\ell_{\mathsf{VOLE}} = \ell + \tau \cdot r \cdot (D-1)$. Notice that also the number of leaves N must be set differently for VOLEitH: this matter is discussed in the next section.

2.5.2 Lifting to a larger field

As said before, VOLEitH requires to lift Δ into a larger field, to achieve security without doing parallel repetitions. Notice that choosing directly q^r to be exponentially large would require the GGM to have an exponential number of leaves, making it impossible to realize in practice. To solve this issue, the main idea is to create multiple VOLE correlations in a smaller field and then concatenate them to obtain one single VOLE correlation over the desired field [3]. Indeed, after τ repetitions of the above procedure, the prover and the verifier know τ (small) VOLE correlations. Denote these by $(\mathbf{u}_i, \mathbf{v}_i)$, with $\mathbf{u}_i \in \mathbb{F}_q^\ell, \mathbf{v}_i \in \mathbb{F}_{q^r}^\ell$, and $(\Delta_i, \mathbf{d}_i)$, with $\Delta_i \in \mathbb{F}_{q^r}, \mathbf{d}_i \in \mathbb{F}_{q^r}^\ell$, for $i \in [\tau]$. By concatenating these VOLE instances, $\mathcal{P}$ and $\mathcal{V}$ form the matrices $\mathbf{U} \in \mathbb{F}_q^{\ell\times\tau}, \mathbf{V}, \mathbf{D}' \in \mathbb{F}_{q^r}^{\ell\times\tau}$ such that

$$\mathbf{D}' = \mathbf{U}\cdot \mathrm{diag}(\boldsymbol{\Delta}) + \mathbf{V}$$

where $\boldsymbol{\Delta} = (\Delta_1, ..., \Delta_\tau)^\top \in \mathbb{F}_{q^r}^\tau$. To obtain a final VOLE correlation with $\mathbf{u} \in \mathbb{F}_q^\ell$, the prover can fix the τ small VOLEs to use the same $\mathbf{u}$, say $\mathbf{u} = \mathbf{u}_1$, by computing the corrections $\mathbf{e}_i = \mathbf{u}_i - \mathbf{u}$ for $i \in [2, \tau]$ and sending them to the verifier. Then, $\mathcal{V}$ can correct its own output by computing

$$\begin{aligned}\mathbf{D} = \mathbf{D}' - [\mathbf{0} \mid \mathbf{e}_2 \mid \cdots \mid \mathbf{e}_\tau] \cdot \mathrm{diag}(\boldsymbol{\Delta}) &= [\mathbf{u} \mid \cdots \mid \mathbf{u}] \cdot \mathrm{diag}(\boldsymbol{\Delta}) + \mathbf{V} \\ &= \mathbf{u} \cdot [1 \mid \cdots \mid 1] \cdot \mathrm{diag}(\boldsymbol{\Delta}) + \mathbf{V} \\ &= \mathbf{u} \cdot \boldsymbol{\Delta}^\top + \mathbf{V}\end{aligned}$$

Finally, viewing $\boldsymbol{\Delta}^\top$ and each row of $\mathbf{V}$ and $\mathbf{D}$ as elements of $\mathbb{F}_{q^{r\tau}}$, the above relation yields a VOLE correlation of length ℓ over $\mathbb{F}_{q^{r\tau}}$ satisfying:

$$\mathbf{d}^* = \mathbf{u} \cdot \Delta^* + \mathbf{v}^*.$$

Since now $\Delta^* \in \mathbb{F}_{q^{r\tau}}$, if the challenges $\gamma_1, \ldots, \gamma_t$ are also sampled from $\mathbb{F}_{q^{r\tau}}$ we recover the formula for the soundness error from Equation (3). Hence, τ can be chosen such that

$$\tau \geq \frac{\lambda + \log_2(D)}{r \log_2(q)}$$

to obtain a sound signature scheme.

Notice that the prover $\mathcal{P}$ can cheat when sending the corrections $\{\mathbf{e}_i\}_{i \in [2,\tau]}$. Indeed, if any $\mathbf{e}_i$ is generated incorrectly, $\mathcal{V}$ will not be able to properly correct its 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 [28]. 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 $\mathbf{H}^{\mathsf{UHF}} \in \mathbb{F}_{q^r}^{m \times \ell_{\mathsf{VOLE}}}$, which is sampled by $\mathcal{V}$ and sent as a challenge to $\mathcal{P}$, who computes

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

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

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

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 7 ($\mathbb{F}_q^\ell$-hiding matrix)**.** A matrix $\mathbf{A} \in \mathbb{F}_{q^r}^{m \times (\ell+h)}$ is $\mathbb{F}_q^\ell$*-hiding* if the distribution of $\mathbf{Av}$ is independent of $\mathbf{v}_{[1..\ell]}$ when $\mathbf{v}_{[\ell+1..\ell+h]} \xleftarrow{\$} \mathbb{F}_q^h$.

Hence, $\mathcal{P}$ and $\mathcal{V}$ need to use VOLE correlations of length $\hat{\ell}_{\mathsf{VOLE}} = (\ell_{\mathsf{VOLE}} + h)$ whose last h entries are needed to perform the consistency check without any leakage. Indeed, by using an $\mathbb{F}_q^{\ell_{\mathsf{VOLE}}}$-hiding hash function $\mathbf{H}^{\mathsf{UHF}} \in \mathbb{F}_{q^r}^{m \times (\hat{\ell}_{\mathsf{VOLE}})}$, the first ℓ_{VOLE} entries of $\mathbf{u}$ will remain hidden to the verifier $\mathcal{V}$. Also, notice that if $\mathbf{H}^{\mathsf{UHF}}$ is $\mathbb{F}_q^{\ell_{\mathsf{VOLE}}}$-hiding, then it is also $\mathbb{F}_{q^r}^{\ell_{\mathsf{VOLE}}}$-hiding: therefore, the first ℓ_{VOLE} rows of $\mathbf{V}$ will also remain hidden to $\mathcal{V}$. In this work, we employ the same consistency check used in FAEST [29] and PERK [9], since it does not depend on the underlying Proof of Knowledge (PoK), and, thus,

we accordingly set $h = \lambda + B$ with $B = 16$. More details about this construction can be found in Appendix C.

2.5.3 Batch Vector Commitment Scheme

Generating the degree-1 polynomials $(p_j)_{j\in[\ell]}$, $(\bar{p}_i)_{i\in[D-1]}$ using GGM trees incurs a significant communication overhead. Indeed, the prover needs to send to the verifier $\log_2(N)$ nodes of the tree, namely the sibling nodes of the nodes on the path from the root to the hidden leaf, together with the commitment to the hidden seed. This is especially true when the τ repetitions of the generation procedure are needed[3]. To solve this issue, [30] introduces a *batch all-but-one vector commitment scheme* (BAVC) which, intuitively, consists of generating a larger tree with $\tau \cdot N$ leaves, instead of τ smaller trees, and then opening all but τ leaves, one per merged smaller tree. This approach turns out to be more efficient because, with high probability, the paths from the root to each unopened leaf merge relatively close to the leaves, thus reducing the number of hidden internal nodes. However, the number of hidden internal nodes strictly depends on the indexes I of the unopened leaves, causing fluctuation in the final signature size. Hence, a threshold T_{open} is fixed to limit the number of internal nodes to be revealed in the opening phase. If T_{open} is exceeded, the computation is aborted.[4] In practice, the value of T_{open} is chosen as a tradeoff between the size of the commitment and the probability of success of the opening phase (which, as it will become clearer later, determines the efficiency of the signing algorithm). The algorithms for the BAVC are deferred to Appendix A.2, while the procedure to generate the correlations from a BAVC is presented in Appendix B.

2.5.4 Smaller challenges through grinding

As done in [4, 29], when applying Fiat-Shamir to the protocol, we use *grinding proof-of-work* when computing the last challenge Δ using a hash function as explained in [30]. In practice, the prover checks whether the last w_{grind} bits of the last challenge are equal to zero: if this is not the case, then the signer increments a counter and recomputes the challenge until this constraint is satisfied. This is performed prior to the opening with abort of the BAVC scheme. The grinding strategy is particularly beneficial because it allows using shorter challenges while also maintaining the desired soundness security level. Indeed, this technique increases the cost of hashing the last challenge by a factor $2^{w_{\mathsf{grind}}}$, increasing the soundness by w_{grind} bits. In particular, in the VOLEitH framework, one should now choose N and τ such that

$$\frac{D}{N^\tau} \cdot \frac{1}{2^{w_{\mathsf{grind}}}} \leq \frac{1}{2^\lambda}$$

[3] Both VOLEitH and TCitH require τ repetitions of the generation procedure: for VOLEitH, this is due to the multiple pairs that are merged into a single correlation, while for TCitH it is due to the parallel repetitions of the whole protocol.

[4] In the non-interactive protocol, the prover checks whether the opening is successful and, if the procedure aborts, the indexes of the unopened leaves are re-sampled.

and, in the TCitH framework, such that

$$\left(\frac{D}{N}\right)^{\tau} \cdot \frac{1}{2^{w_{\mathsf{grind}}}} \leq \frac{1}{2^{\lambda}}.$$

When integrating this improvement, the length of the PRG output changes again for both the VOLEitH and the TCitH frameworks, since they now have to take into account the w_{grind} parameter.

3 Polynomial modelling for SEP

As mentioned in Section 2, to build a PIOP we need to find a polynomial system of equations F_{SEP} that models a given SEP instance $(\mathscr{C}, \mathscr{C}')$ and such that π is a solution of it if and only if $F_{\mathsf{SEP}}(\pi) = 0$.

Hence, in this section, we describe two distinct polynomial modellings for *in-the-Head* signatures based on SEP. The first one adapts the third modelling from [14] to work with instances of SEP. Instead, the second one employs the modelling for permutation matrices introduced in [31, Section 5.3]. Notice that the modellings presented below are correct for instances of SEP defined over fields of characteristic 2.

3.1 First modelling

Throughout this section, we will write $d = \lceil \log_2(n) \rceil$. We will denote by $\mathcal{B}^j \in \mathbb{F}_2^d$ the binary representation of an integer $j \in [0, n-1]$, and by $\overline{\mathcal{B}^j}$ its bitwise complement.

Observe that, given $\mathbf{b}^1, \mathbf{b}^2 \in \mathbb{F}_2^d$, $\mathbf{b}^1 + \mathbf{b}^2 = (1, ..., 1)^\top$ if and only if $\mathbf{b}^2 = \overline{\mathbf{b}^1}$. As a consequence, the product of the entries of $\mathbf{b}^1 + \mathbf{b}^2$ is equal to 1 if and only if $\mathbf{b}^2 = \overline{\mathbf{b}^1}$. Given an integer $\mathsf{pos} \in [0, n-1]$, let $\mathcal{B}^{\mathsf{pos}} = (p_0, \ldots, p_{d-1}) \in \mathbb{F}_2^d$. Then, the vector $\mathbf{z} = (z_0, \ldots, z_{n-1}) \in \mathbb{F}_2^n$ defined as

$$z_j := \prod_{k=0}^{d-1} (p_k + \overline{\mathcal{B}^j}_k), \quad \forall j \in [0, n-1] \tag{4}$$

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 $\mathbf{z}$ given the position of the 1. With this idea, we can represent a permutation matrix $\mathbf{P}$ by n integers $\mathsf{pos}_0, ..., \mathsf{pos}_{n-1} \in [0, n-1]$, which correspond to the positions of the non-zero entry in each row of $\mathbf{P}$: then, we can reconstruct each row using the procedure shown above. We thus employ $n \cdot d$ variables $\{p_{i,j}\}_{i \in [0,n-1], j \in [0,d-1]}$ to represent the permutation matrix $\mathbf{P}$ by means of $\mathsf{pos}_0, \ldots, \mathsf{pos}_{n-1}$ by setting $\mathcal{B}^{\mathsf{pos}_i} = (p_{i,0}, \ldots, p_{i,d-1})$ $\forall i \in [0, n-1]$. Then, F_{SEP} is realized by the following sets of polynomials:

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

$$\mathcal{S}_{\mathsf{col}} = \left\{ \left(\sum_{i=0}^{n-1} Z(i,j) \right) - 1 = 0 \mid j \in [0, n-1] \right\} \tag{5}$$

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

$$Z(i,j) := \prod_{m=0}^{d-1} \left(p_{i,m} + \overline{\mathcal{B}^j}_m \right). \tag{6}$$

By construction, for each $i \in [0, n-1]$ 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 (4). Indeed, for all $i \in [0, n-1]$, $Z(i,j) = 1$ if $j = \mathsf{pos}_i$, whereas $Z(i,j) = 0$ whenever $j \neq \mathsf{pos}_i$.

– $k' \cdot (n-k)$ equations of degree d imposing that the secret $\mathbf{P}$ satisfies the equation $\mathbf{G}'\mathbf{P}\mathbf{H}^\top = \mathbf{0}_{k',n-k}$

$$\mathcal{S}_{\mathsf{eq}} = \left\{ \sum_{m=0}^{n-1} \left(\mathbf{G}'_{i,m} \cdot \sum_{\ell=0}^{n-1} Z(m,\ell) \cdot \mathbf{H}_{j,\ell} \right) = 0 \mid i \in [0, k'-1], j \in [0, n-k-1] \right\} \tag{7}$$

Overall, the sets $\mathcal{S}_{\mathsf{col}}, \mathcal{S}_{\mathsf{eq}}$ form a system of $n + k' \cdot (n-k)$ polynomial constraints of degree at most $D = d$ in $n \cdot d$ variables.

Remark 3. The modelling presented above is correct only for instances of SEP defined over $\mathbb{F}_{2^m}$, for some $m \geq 1$. This is true because in odd characteristic $1+1 \neq 0$: for this reason, as shown in [14], a higher-degree relation is needed to reconstruct elementary vectors as in Equation (4). Briefly, over $\mathbb{F}_q$ with q odd, $\mathcal{B}^{\mathsf{pos}} \in \{0,1\}^d \subset \mathbb{F}_q^d$ and, in Equation (4), the sum is replaced by the subtraction modulo q: hence, $z_j \in \{-1, 1\}$ if $j = \mathsf{pos}$ and $z_j = 0$ otherwise. To guarantee that $z_{\mathsf{pos}} = 1$, we modify Equation (6) as follows

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

In addition to this, the modelling for the case of odd characteristic requires an additional set of equations

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

to ensure that the variables $(p_{i,j})_{i \in [0,n-1], j \in [0,d-1]}$ have values in $\{0,1\}$. Therefore, the overall system will have degree $D = 2d$, resulting in a less efficient signature scheme. For this reason, the instances proposed in this work are defined over binary fields $\mathbb{F}_{2^m}, m \geq 1$.

Proposition 1. *Let $(\mathbf{G}', \mathbf{H}) \in \mathbb{F}_q^{k' \times n} \times \mathbb{F}_q^{(n-k) \times n}$ be an instance of SEP defined over a field of characteristic 2. A permutation matrix $\mathbf{P} \in \mathbb{F}_2^{n \times n}$ is a solution for it if and only if it satisfies the set of equations $\mathcal{S}_{\mathsf{col}}, \mathcal{S}_{\mathsf{eq}}$.*

Proof We first show that a solution for the SEP problem satisfies the system of equations. Let $\mathbf{P}$ be a permutation matrix. For all $0 \leq i \leq n-1$ let $\mathbf{p}_i$ the binary expansion of the non-zero

position of the i-th row. From the observation above, they also satisfy $\mathcal{S}_{\mathsf{col}}$. By construction, a solution of SEP is a solution for $\mathbf{G}'\mathbf{P}\mathbf{H}^\top = \mathbf{0}$, thus $\mathbf{P}$ solves the system.

We now do the converse direction. Let $\{\mathbf{p}_i\}_{i\in\{0,\dots,n-1\}}$ be a solution for the system. Since they are binary vectors each of them represents an integer in $\{0,...,n-1\}$. Since they are solutions of $\mathcal{S}_{\mathsf{col}}$ we have that each integer in $\{0,...,n-1\}$ 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 $\mathbf{P}^*$. Finally, they also satisfy the set $\mathcal{S}_{\mathsf{eq}}$, thus $\mathbf{P}^*$ is a solution of the given SEP instance. □

3.2 Second modelling

In [31], the author introduces a family of representations of elementary vectors to improve the signature size of VOLE-in-the-Head schemes employing these vectors. This is done by encoding elementary vectors using shorter vectors having a fixed weight and using their support to identify the non-zero coordinate of the desired elementary vector.

In particular, [31] introduces these encodings to improve the sizes of SDitH [4] and the VOLEitH signature based on PKP presented in [13] (referred to as *PKPitH* in [31]). In this section, we briefly recall the family of modellings introduced in [31], and then adapt the construction for PKPitH to our instances of SEP. We notice that this construction is defined for instances of the problems over $\mathbb{F}_2$ and its extensions.

3.2.1 The construction from [31]

Let μ, δ be positive integers such that $\mu \geq \delta$. Let $\mathcal{E}_\delta(\mu) \subseteq \mathbb{F}_2^\mu$ be the set of all weight-δ vectors in $\mathbb{F}_2^\mu$. Let $S_1, \dots, S_{\binom{\mu}{\delta}} \subseteq [\mu]$ be the subsets of size δ of $[\mu]$. Finally, given $\mathbf{v} \in \mathcal{E}_\delta(\mu),\ \mathbf{v} = (v_1, \dots, v_\mu)$, define the map $\mathrm{prod}_\delta(\cdot)$ as

$$\mathrm{prod}_\delta(\mathbf{v}) := \left(\prod_{j\in S_1} v_j, \dots, \prod_{j\in S_{\binom{\mu}{\delta}}} v_j \right) \tag{8}$$

Notice that, since $\mathrm{wt}(\mathbf{v}) = \delta$, then $\mathrm{wt}(\mathrm{prod}_\delta(\mathbf{v})) = 1$ because, for all $i \in \left[\binom{\mu}{\delta}\right]$, $\prod_{j\in S_i} v_j = 1$ if and only if $S_i = \mathrm{supp}(\mathbf{v})$. In other words, $\mathrm{prod}_\delta(\mathbf{v})$ is an elementary vector of length $\binom{\mu}{\delta}$. To obtain an elementary vector $\mathbf{z} = (z_1, \dots, z_n)$ of the desired length n, it suffices to choose μ and δ such that $\binom{\mu}{\delta} \geq n$ and compute

$$\mathbf{z} = \mathbf{M} \cdot \mathrm{prod}_\delta(\mathbf{v}),$$

where $\mathbf{M} \in \mathbb{F}_2^{n\times\binom{\mu}{\delta}}$ is chosen such that each column of $\mathbf{M}$ has weight 0 or 1 and $\mathbf{M}$ has full rank or, equivalently, $\mathbf{M}$ has no zero rows.

Notice that, since $\mathrm{prod}_\delta(\mathbf{v})$ has weight 1, $\mathrm{wt}(\mathbf{z}) \in \{0, 1\}$. Therefore, this encoding can be used to represent elementary vectors if we impose the following constraints:

- $\mathrm{wt}(\mathbf{v}) = \delta$
- $\mathrm{wt}(\mathbf{M} \cdot \mathrm{prod}_\delta(\mathbf{v})) = 1$.

To represent these constraints by polynomial equations, let $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_\mu)$ be a vector of variables representing a weight-δ vector $\mathbf{v}$. The first constraint can then be represented by equations of the form

$$\nu_{j_1} \cdots \nu_{j_{\delta+1}} = 0, \ j_1 < j_2 < \cdots < j_{\delta+1} \tag{9}$$

$$\sum_{j_1 < \cdots < j_\delta} \nu_{j_1} \cdots \nu_{j_\delta} = 1 \tag{10}$$

Instead, for the second constraint, notice that for all $i \in [n]$

$$z_i = \sum_{\substack{1 \le j \le \binom{\mu}{\delta} \\ \mathbf{M}_{i,j}=1}} (\mathrm{prod}_\delta(\mathbf{v}))_j.$$

Therefore, since $\mathrm{prod}_\delta(\mathbf{v})$ has weight 1, z_i will be equal to 1 whenever there is a $j \in [\binom{\mu}{\delta}]$ such that $\mathbf{M}_{i,j} = (\mathrm{prod}_\delta(\mathbf{v}))_j = 1$. Equivalently, this relation can be expressed as

$$\mathrm{wt}(\mathbf{M}_{:,j}) = 0 \implies (\mathrm{prod}_\delta(\mathbf{v}))_j = 0$$

Again, this is equivalent to

$$\sum_{\substack{1 \le j \le \binom{\mu}{\delta} \\ \mathrm{wt}(\mathbf{M}_{:,j})=0}} (\mathrm{prod}_\delta(\mathbf{v}))_j = 0$$

The latter provides the following polynomial equation for the second constraint

$$\sum_{\substack{1 \le j \le \binom{\mu}{\delta} \\ \mathrm{wt}(\mathbf{M}_{:,j})=0}} (\mathrm{prod}_\delta(\boldsymbol{\nu}))_j = 0 \tag{11}$$

As in the previous modelling, we need variables to represent n elementary vectors of length n which will be the rows of the $n \times n$ secret permutation matrix $\mathbf{P}$. Hence, we employ n vectors of μ variables $\boldsymbol{\nu}^{(1)}, \ldots, \boldsymbol{\nu}^{(n)}$ to represent $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(n)} \in \mathcal{E}_\delta(\mu)$ and apply the constraints in Equations (9) to (11) to each one.

Finally, to ensure that the matrix obtained by stacking these vectors is a permutation matrix, n additional constraints are needed to ensure that each column has weight equal to 1:

$$\sum_{i=1}^{n} (\mathbf{M} \cdot \mathrm{prod}_\delta(\boldsymbol{\nu}^{(i)}))_j = 1, \ j = 1, \ldots, n \tag{12}$$

The latter equations still have degree equal to δ. Moreover, notice that Equations (9) to (12) are instance-independent and simply enforce the constraints to represent an $n \times n$ permutation matrix. Then, to obtain a polynomial modelling for an instance $(\mathbf{G}', \mathbf{H})$ of SEP, we just need to notice that, now, $\mathbf{P} = \left[(\mathbf{M} \cdot \mathrm{prod}_\delta(\boldsymbol{\nu}^{(i)}))_j\right]_{i,j \in [n]}$ and

plug this representation into the relation $\mathbf{G}'\mathbf{P}\mathbf{H}^\top = \mathbf{0}_{k',n-k}$ to obtain the polynomial equations

$$\mathcal{S}_{\mathsf{eq}} = \left\{ \sum_{m=1}^{n} \left(\mathbf{G}'_{i,m} \cdot \sum_{\ell=1}^{n} (\mathbf{M} \cdot \mathrm{prod}_\delta(\boldsymbol{\nu}^{(m)}))_\ell \cdot \mathbf{H}_{j,\ell} \right) = 0 \mid i \in [k'], j \in [n-k] \right\} \quad (13)$$

Remark 4. The choice of μ, δ can be refined by partitioning the vectors $\mathbf{v}^{(i)} \in \mathcal{E}_\delta(\mu)$ into $t > 1$ subvectors $\mathbf{v}^{(i,1)} \in \mathbb{F}_2^{\mu_1}, \ldots, \mathbf{v}^{(i,t)} \in \mathbb{F}_2^{\mu_t}$ of lengths $\mu_1, \ldots, \mu_t$ such that

$$\begin{cases} \mathbf{v}^{(i)} = (\mathbf{v}^{(i,1)} || \cdots || \mathbf{v}^{(i,t)}) \\ \sum_{j=1}^{t} \mu_j = \mu \\ \sum_{j=1}^{t} \mathrm{wt}(\mathbf{v}^{(i,j)}) = \delta \end{cases}$$

Indeed, in this case the relation $\binom{\mu}{\delta} \geq n$ can be replaced by a tighter relation depending on the values of $\mu_1, \ldots, \mu_t$ and on how the weight δ is distributed among the subvectors. Moreover, the distribution of the weight among the subvectors affects also the polynomial constraints in Equations (9) and (10). The details about this improvement become clearer by looking at an example, such as that presented in the following section.

3.2.2 Concrete instance of the modelling

To obtain a concrete instance of the modelling described in the previous section, we adapt the instance described in [31, Section 5.3] for PKPitH to work with our parameter suites.

As stated in the following sections, for our parameter suites the lengths n of the codes are included in the interval $[74, 95]$. Let $t = 2$, $(\mu_1, \mu_2) = (4, 6)$ and $\delta = 3$. As displayed in the previous section, for each $i \in [n]$ we represent an elementary vector $\mathbf{z}_i \in \mathbb{F}_2^n$ by a vector $\mathbf{v}^{(i)} \in \mathcal{E}_\delta(\mu)$. In this section, however, we restrict ourselves to a specific subset $\mathcal{V} \subseteq \mathcal{E}_\delta(\mu)$ defined as follows

$$\mathcal{V} = \Big\{ \mathbf{v} \in \mathcal{E}_\delta(\mu) \mid \mathbf{v}^{(i)} = (\mathbf{v}^{(i,1)} || \mathbf{v}^{(i,2)}),\ \mathbf{v}^{(i,1)} \in \mathbb{F}_2^{\mu_1},\ \mathbf{v}^{(i,2)} \in \mathbb{F}_2^{\mu_2},\ \{\mathrm{wt}(\mathbf{v}^{(i,1)}), \mathrm{wt}(\mathbf{v}^{(i,2)})\} = \{1, 2\} \Big\}$$

As a sanity check, notice that $|\mathcal{V}| = \binom{4}{1}\binom{6}{2} + \binom{4}{2}\binom{6}{1} = 96$ which is larger than any of our proposed code-lengths n. Hence, for any of our parameter suites, each $\mathbf{z}_i \in \mathcal{E}_1(n)$ can be represented using at least one vector in $\mathcal{V}$.

For each $i \in [n], j \in \{1, 2\}$, let $\boldsymbol{\nu}^{(i,j)} = (\nu_1^{(i,j)}, \ldots, \nu_{\mu_j}^{(i,j)})$ be the vector of variables corresponding to $\mathbf{v}^{(i,j)}$. With this choice of μ, δ, t, Equations (9) and (10) are replaced by the following equations

$$\nu_{l_1}^{(i,j)} \nu_{l_2}^{(i,j)} \nu_{l_3}^{(i,j)} = 0,\ 1 \leq l_1 < l_2 < l_3 \leq \mu_j,\ j \in \{1, 2\},\ i \in [n] \quad (14)$$

$$\sum_{1\le l_1<l_2\le \mu_j} (\nu_{l_1}^{(i,j)}\nu_{l_2}^{(i,j)}) + \sum_{l=1}^{\mu_j} \nu_l^{(i,j)} = 1,\ j = 1,2,\ i \in [n] \tag{15}$$

$$\sum_{l=1}^{\mu_1} \nu_l^{(i,1)} + \sum_{l=1}^{\mu_2} \nu_l^{(i,2)} = 1, i \in [n] \tag{16}$$

Equation (14) ensures that $\mathrm{wt}(\nu^{(i,j)}) < 3$ for each $i \in [n]$ and $j = 1, 2$. Equation (15) ensures that $\mathrm{wt}(\nu^{(i,j)}) \in \{1, 2\}$ for each $i \in [n]$ and $j = 1, 2$. Equation (16) ensures that $\{\mathrm{wt}(\nu^{(i,1)}), \mathrm{wt}(\nu^{(i,2)})\} = \{1, 2\}$ for each $i \in [n]$. Hence, the solutions to these equations are elements of $\mathcal{V}$.

Let $\mathcal{S}_{\mathsf{row}}$ denote the set containing all polynomial equations in Equations (14) to (16). Similarly, let $\mathcal{S}_{\mathsf{col}}$ denote the polynomial equations from Equation (12), and let $\mathcal{S}_{\mathsf{eq}}$ be as in Equation (13). Then, the sets $\mathcal{S}_{\mathsf{row}}, \mathcal{S}_{\mathsf{col}}, \mathcal{S}_{\mathsf{eq}}$ form a system of

$$\underbrace{k' \cdot (n-k)}_{\mathcal{S}_{\mathsf{eq}}} + \underbrace{n}_{\mathcal{S}_{\mathsf{col}}} + \underbrace{n \cdot \left(3 + \binom{\mu_1}{3} + \binom{\mu_2}{3}\right)}_{\mathcal{S}_{\mathsf{row}}}$$

equations of degree D at most $\delta = 3$.

Proposition 2. *Let $(\mathbf{G}', \mathbf{H}) \in \mathbb{F}_q^{k' \times n} \times \mathbb{F}_q^{(n-k)\times n}$ be an instance of SEP defined over a field of characteristic 2. A permutation matrix $\mathbf{P} \in \mathbb{F}_2^{n\times n}$ is a solution for it if and only if it satisfies the set of equations $\mathcal{S}_{\mathsf{row}}, \mathcal{S}_{\mathsf{col}}, \mathcal{S}_{\mathsf{eq}}$.*

The proof of the latter proposition is analogous to that of Proposition 1 and follows from the construction of the system of equations.

Remark 5. Similarly to the previous model, this modelling is only correct for instances of SEP defined over fields of characteristic 2. It can be easily adapted to work for a generic field by adding equations analogous to those in $\mathcal{S}_{\mathsf{pos}}$ to impose that the variables $\boldsymbol{\nu}^{(1)}, \ldots, \boldsymbol{\nu}^{(n)}$ take values in $\{0, 1\}$.

4 In-the-Head Signatures from SEP

In this section, we present our digital signature schemes based on SEP using the VOLEitH and the TCitH frameworks. Briefly, the signature scheme is defined by the following algorithms:

- **Key Generation:** The signer picks a random code $\mathscr{C}$ with parity-check matrix $\mathbf{H}$ and a random permutation matrix $\mathbf{P}$. Then, the signer selects a subcode $\bar{\mathscr{C}}$ of $\mathscr{C}$. Let $\bar{\mathbf{G}}$ be the generator matrix of $\bar{\mathscr{C}}$. The private key is $\mathbf{P}$, while the public key is $(\mathbf{G}', \mathbf{H})$, where $\mathbf{G}' = \bar{\mathbf{G}}\mathbf{P}^{-1}$ as generator matrix[5].
- **Signature Algorithm:** The signer transforms the public key into a system of equations as per Section 3, and exploits the knowledge of $\mathbf{P}$ to obtain a solution $\mathbf{w}$

[5]Notice that instead of saving $\mathbf{H}$, one can simply store seed and use it to recompute $\mathbf{H}$.

for it. The signature algorithm is obtained by applying the Fiat-Shamir transform to the PIOP scheme explained in Section 2.5, using either TCitH or VOLEitH to ensure soundness.
– **Verification Algorithm:** The verification is the same as the interactive version of PIOP, with the additional checks required by either TCitH or VOLEitH.

The complete algorithms for both the VOLEitH and the TCitH versions of the signature are shown respectively in Algorithms 5 and 6 and Algorithms 7 and 8.

4.1 Common algorithms and subroutines

The TCitH and the VOLEitH approaches share the same key generation procedure, as it solely relies on the underlying problem. Such procedure is described in Algorithm 3.

Algorithm 3: Protocol $\mathsf{SETH}_{\mathsf{KeyGen}}(\lambda, m, n, k', k, \tau, r, w_{\mathsf{grind}})$

Input: Security parameter λ, extension degree m, code parameters n, k, k', protocol parameters $\tau, r, w_{\mathsf{grind}}$.
Output: Pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$;
$\mathbf{H} \xleftarrow{\$,\mathsf{seed}} \mathbb{F}_{2^m}^{(n-k)\times n}$ with $\mathrm{rank}(\mathbf{H}) = n-k$. Let $\mathscr{C} = \ker(\langle \mathbf{H} \rangle)$;
$\mathbf{P} \xleftarrow{\$} \mathsf{S}_n$;
Pick uniformly at random k' linearly independent vectors $(\mathbf{c}_1, ..., \mathbf{c}_{k'})$ from $\mathscr{C}$. Let $\bar{\mathbf{G}}$ be the $k' \times n$ matrix having them as rows. Set $\mathbf{G}' = \bar{\mathbf{G}}\mathbf{P}^{-1}$;
Set $\mathsf{pk} = (\mathbf{G}', \mathbf{H})$, $\mathsf{sk} = \mathbf{P}$;
Set $\mathsf{pp} = (\lambda, m, n, k', k, \tau, r, w_{\mathsf{grind}})$.

Algorithm 4: Subroutines for non-interactive version of PIOP

$\mathsf{NIZK.Prove}(\mathbf{w}, (f_1, \dots, f_t), (v_j)_{j\in[\ell]}, (\bar{u}_{j,i}, \bar{v}_{j,i})_{j\in[\rho], i\in[D-1]}, \rho, \mathbf{\Gamma})$:
for $j \in [\ell]$ **do**
  $p_j(X) = w_j X + v_j$;
for $j \in [t]$ **do**
  $\bar{f}_j(X) = X^D f_j(\frac{p_1(X)}{X}, \dots, \frac{p_\ell(X)}{X})$;
for $j \in [\rho]$ **do**
  $\mathrm{mask}_j(X) = \sum_{i=1}^{D-1} (\bar{u}_{j,i} X + \bar{v}_{j,i}) X^{i-1}$;
  $\bar{F}_j(X) = \sum_{i=1}^{t} \mathbf{\Gamma}_{j,i} \bar{f}_i(X)$;
  $F_{\mathsf{masked},j}(X) = \bar{F}_j(X) + \mathrm{mask}_j(X)$;
  Set $(a_{j,0}, \dots, a_{j,D-1}) = \mathrm{Coefficients}(F_{\mathsf{masked},j})$;
return $((a_{j,0}, \dots, a_{j,D-1}))_{j\in[\rho]}$

$\mathsf{NIZK.Verify}(\Delta, (f_1, \dots, f_t), (d_j)_{j\in[\ell]}, (\bar{d}_{j,i})_{j\in[\rho], i\in[D-1]}, ((a_{j,i})_{i\in[D-1]})_{j\in[\rho]}, \rho, \mathbf{\Gamma})$:
for $j \in [t]$ **do**
  $\bar{f}_j(\Delta) = \Delta^D f_j(\frac{d_1}{\Delta}, \dots, \frac{d_\ell}{\Delta})$;
for $j \in [\rho]$ **do**
  $\mathrm{mask}_j(\Delta) = \sum_{i=1}^{D-1} \bar{d}_{j,i} \Delta^{i-1}$;
  let $(\gamma_1, \dots, \gamma_t) = (\mathbf{\Gamma}_{j,1}, \dots, \mathbf{\Gamma}_{j,t})$;
  $\bar{F}_j(\Delta) = \sum_{i=1}^{t} \gamma_i \bar{f}_i(\Delta)$;
  $F_{\mathsf{masked},j}(\Delta) = \bar{F}_j(\Delta) + \mathrm{mask}_j(\Delta)$;
  $\tilde{a}_{j,0} = F_{\mathsf{masked},j}(\Delta) - \sum_{i=1}^{D-1} a_{j,i} \cdot \Delta^i$
return $(\tilde{a}_{j,0})_{j\in[\rho]}$

Moreover, we define the sub-routines NIZK.Prove and NIZK.Verify in Algorithm 4 to model the non-interactive version of the PIOP described in Section 2.5, which will be used by both approaches to obtain the signature scheme. Since the difference in the two techniques lies in the number ρ of t-tuples generated by the verifier, we add this value as input to the procedures. We recall that D is the degree of the polynomial system, according to Section 3.1 or Section 3.2.

Other additional functions for conversion between field elements and bit strings are contained in Appendix D.

Algorithm 5: Protocol $\mathsf{SETH}_{\mathsf{Sign-VOLE}}(\mathsf{m}, \mathsf{sk}, \mathsf{pk}, \mathsf{pp})$

Input: Message m to be signed; public key $\mathsf{pk} = (\mathbf{G}', \mathbf{H})$ with $\mathbf{G}' \in \mathbb{F}_{2^m}^{k' \times n}$ and $\mathbf{H} \in \mathbb{F}_{2^m}^{(n-k)\times n}$; secret key $\mathsf{sk} = \mathbf{P} \in \mathsf{S}_n$ such that $\mathbf{G}'\mathbf{P}\mathbf{H}^\top = \mathbf{0}$.
Output: Signature σ for the message m.
Compute the degree-D polynomial equations $\{f_1, \ldots, f_t\} \subset \mathbb{F}_{2^m}[X_1, \ldots, X_\ell]$ from pk and a solution $\mathbf{w} \in \mathbb{F}_2^\ell$ from sk;

$\mathbf{r} \xleftarrow{\$} \{0,1\}^\lambda$;
$\mu \leftarrow \mathsf{H}(\mathsf{pk}||\mathsf{m})$;
$\mathsf{iv} \leftarrow \mathsf{H}(\mathsf{sk}||\mu||\mathbf{r})$;
$(\mathsf{com}, \mathsf{decom}, \mathbf{e}_2, ..., \mathbf{e}_\tau, \mathbf{u}, \mathbf{V}) \leftarrow \mathsf{VOLE.Commit}(\mathbf{r}, \mathsf{iv}, \ell + D \cdot \lambda + 16)$;
$\mathsf{ch}_1 \leftarrow \mathsf{H}(\mu||\mathsf{com}||\mathbf{e}_2||...||\mathbf{e}_\tau||\mathsf{iv})$;
$\widetilde{\mathbf{u}} \leftarrow \mathsf{VOLEHash}(\mathsf{ch}_1, \mathbf{u})$;
$\widetilde{\mathbf{V}} \leftarrow \mathsf{VOLEHash}(\mathsf{ch}_1, \mathbf{V})$; // Hash each column individually
$\mathbf{t} \leftarrow \mathbf{w} - \mathbf{u}_{[1..\ell]}$;
$\mathsf{ch}_2 \leftarrow \mathsf{H}(\mathsf{ch}_1||\widetilde{\mathbf{u}}||\widetilde{\mathbf{V}}||\mathbf{t})$;
$\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_t) \leftarrow \mathsf{ExpandGamma}(\mathsf{ch}_2)$;
for $i \in [\ell + (D-1) \cdot \lambda]$ **do**
  $v_i \leftarrow \mathsf{ToField}((\mathbf{V}_{i,1}, \ldots, \mathbf{V}_{i,\tau}))$;
for $i \in [D-1]$ **do**
  $\overline{u}_i \leftarrow \mathsf{ToField}((u_{\ell+(i-1)\cdot\lambda+1}, \ldots, u_{\ell+i\cdot\lambda}), \lambda)$;
  $\overline{v}_i \leftarrow \mathsf{ToField}((v_{\ell+(i-1)\cdot\lambda+1}, \ldots, v_{\ell+i\cdot\lambda}), \lambda)$;
$(a_0, \ldots, a_{D-1}) \leftarrow \mathsf{NIZK.Prove}(\mathbf{w}, (f_1, \ldots, f_t), (v_i)_{i\in[\ell]}, (\overline{u}_i, \overline{v}_i)_{i\in[D-1]}, 1, \boldsymbol{\gamma})$;
$\mathsf{ctr} = -1$;
repeat
  $\mathsf{ctr} = \mathsf{ctr} + 1$;
  $\mathsf{ch}_3 \leftarrow \mathsf{H}(\mathsf{ch}_2||a_0||\cdots||a_{D-1}||\mathsf{ctr})$;
  if $\mathsf{ch}_3[\lambda - w_{\mathsf{grind}} + 1, \lambda] \neq 0^{w_{\mathsf{grind}}}$ **then**
    continue;
  $I \leftarrow \mathsf{ComputeDelta}(\mathsf{ch}_3)$;
  $\mathsf{decom}_I \leftarrow \mathsf{BAVC.Open}(\mathsf{decom}, I, \mathsf{iv})$;
until $\mathsf{decom}_I \neq \perp$;
return $\sigma \leftarrow \big((\mathbf{e}_i)_{i\in[2,\tau]}, \widetilde{\mathbf{u}}, \mathbf{t}, (a_j)_{j\in[D-1]}, \mathsf{decom}_I, \mathsf{ch}_3, \mathsf{iv}, \mathtt{ctr}\big)$

4.2 VOLE-in-the-Head Signature

As already mentioned in the previous section, Algorithms 5 and 6 depict the signing and verification algorithms for the VOLEitH version of our signature scheme. Additionally to the subprocedures described previously, the two algorithms employ the following subprocedures to extract the challenges from a seed as depicted below:

– $\mathsf{ExpandGamma}(\mathsf{ch})$: Given a seed ch, employs an extendable output function (XOF) to return a vector $(\gamma_1, \ldots, \gamma_t) \in \mathbb{F}_{2^\lambda}^t$;

Algorithm 6: Protocol $\mathsf{SETH}_{\mathsf{Verify-VOLE}}(\sigma, \mathsf{m}, \mathsf{pk}, \mathsf{pp})$

Input: Signature σ, signed message m, public key pk.
Output: Accept or reject.
Compute the degree-D polynomial equations $\{f_1, \ldots, f_t\} \subset \mathbb{F}_{2^m}[X_1, \ldots, X_\ell]$ from pk;
parse $\sigma = \big((\mathbf{e}_i)_{i\in[2,\tau]}, \tilde{\mathbf{u}}, \mathbf{t}, (a_j)_{j\in[D-1]}, \mathsf{decom}_I, \mathsf{ch}_3, \mathsf{iv}, \mathtt{ctr}\big)$;
if $\mathsf{ch}_3[\lambda - w_{\mathsf{grind}} + 1, \lambda] \neq 0^{w_{\mathsf{grind}}}$ **then**
  reject;
$\mu \leftarrow \mathsf{H}(\mathsf{pk}||\mathsf{m})$;
$I = (\Delta_1, \ldots, \Delta_\tau) \leftarrow \mathsf{ComputeDelta}(\mathsf{ch}_3)$;
$(\overline{\mathsf{com}}, \mathbf{D}) \leftarrow \mathsf{VOLE.Reconstruct}(I, \mathsf{decom}_I, \mathbf{e}_2, \ldots, \mathbf{e}_\tau, \mathsf{iv}, \ell + D \cdot r \cdot \tau + 16)$;
$\overline{\mathsf{ch}_1} \leftarrow \mathsf{H}(\mu||\overline{\mathsf{com}}||\mathbf{e}_2|| \ldots ||\mathbf{e}_\tau||\mathsf{iv})$;
$\widetilde{\mathbf{D}} \leftarrow \mathsf{VOLEHash}(\overline{\mathsf{ch}_1}, \mathbf{D})$; // Hash each column individually
$\widetilde{\mathbf{V}} = \widetilde{\mathbf{D}} - \tilde{\mathbf{u}} \cdot [1| \cdots |1] \cdot \mathrm{diag}((\Delta_1, \ldots, \Delta_\tau))$;
$\overline{\mathsf{ch}_2} \leftarrow \mathsf{H}(\overline{\mathsf{ch}_1}||\tilde{\mathbf{u}}||\widetilde{\mathbf{V}}||\mathbf{t})$;
$\overline{\gamma} \leftarrow \mathsf{ExpandGamma}(\overline{\mathsf{ch}_2})$;
$\Delta^* \leftarrow \mathsf{ToField}((\Delta_1, \ldots, \Delta_\tau), \lambda)$;
for $i \in [\ell + (D-1) \cdot \lambda]$ **do**
  $d_i \leftarrow \mathsf{ToField}((\mathbf{D}_{i,1}, \ldots, \mathbf{D}_{i,\tau}))$;
for $i \in [D-1]$ **do**
  $\overline{d}_i \leftarrow \mathsf{ToField}((d_{\ell+(i-1)\cdot\lambda+1}, \ldots, d_{\ell+i\cdot\lambda}), \lambda)$;
$\overline{a_0} \leftarrow \mathsf{NIZK.Verify}(\Delta^*, (f_1, \ldots, f_t), (d_j)_{j\in[\ell]}, (\overline{d}_i)_{i\in[D-1]}, (a_i)_{i\in[D-1]}, 1, \overline{\gamma})$;
$\overline{\mathsf{ch}_3} \leftarrow \mathsf{H}(\overline{\mathsf{ch}_2}||\overline{a_0}|| \cdots ||a_{D-1}||\mathtt{ctr})$;
Accept the signature if $\overline{\mathsf{ch}_3} = \mathsf{ch}_3$, otherwise reject.

– $\mathsf{ComputeDelta}(\mathsf{ch})$: Given a challenge ch of size $\tau \cdot r + w_{\mathsf{grind}}$, it discards the trailing w_{grind} bits equal to 0 and returns a tuple of τ elements of $\mathbb{F}_{2^r}$.

$$|\sigma_{\mathsf{VOLE}}| = \underbrace{\lambda}_{\mathsf{iv}} + \underbrace{2 \cdot \lambda \cdot \tau + \lambda \cdot T_{\mathsf{open}}}_{\mathsf{decom}_I} + \underbrace{\tau \cdot \hat{\ell}_{\mathsf{VOLE}}}_{\mathbf{t},\ \text{NIZK output},\tilde{\mathbf{u}}} + \underbrace{\tau \cdot r + w_{\mathsf{grind}}}_{\mathsf{ch}_3} + \underbrace{32}_{\mathtt{ctr}} \tag{17}$$

4.3 TC-in-the-Head Signature

Algorithms 7 and 8 display the signing and verification algorithms for the TCitH version of our signature scheme. As explained in Section 2.5, the main difference is that, instead of lifting the polynomials into an exponentially large extension of $\mathbb{F}_2$, the protocol achieves the desired security level by repeating the protocol in parallel τ times. Additionally to the NIZK protocols, the signing and verification algorithms employ the following procedures to expand challenges:

– $\mathsf{ExpandGamma}(\mathsf{ch})$: Given a seed ch, it employs an extendable output function (XOF) to return a matrix $\mathbf{\Gamma} \in \mathbb{F}_{2^\lambda}^{t\times\rho}$, whose rows will be used as the coefficients of the first challenge of the PIOP in each of the ρ repetitions of the first "challenge-response" round of the PIOP;
– $\mathsf{ComputeDelta}(\mathsf{ch})$: Behaves as in the VOLEitH framework.

The size in bits of the signature obtained using TCitH is given by the following formula

$$|\sigma_{\mathsf{TC}}| = \underbrace{\lambda}_{\mathtt{iv}} + \underbrace{\lambda T_{\mathsf{open}} + 2\lambda\tau}_{\mathsf{decom}_I} + \underbrace{\tau \cdot r + w_{\mathsf{grind}}}_{\mathsf{ch}_3} + \underbrace{\tau \cdot \ell_{\mathsf{TC}}}_{\mathbf{t}\ \text{and NIZK output}} + \underbrace{32}_{\mathtt{ctr}} \tag{18}$$

Algorithm 7: Protocol $\mathsf{SETH}_{\mathsf{Sign-TC}}(\mathsf{m}, \mathsf{sk}, \mathsf{pk}, \mathsf{pp})$

Input: Message m to be signed; public key $\mathsf{pk} = (\mathbf{G}', \mathbf{H})$ with $\mathbf{G}' \in \mathbb{F}_{2^m}^{k' \times n}$ and $\mathbf{H} \in \mathbb{F}_{2^m}^{(n-k) \times n}$; secret key $\mathsf{sk} = \mathbf{P} \in \mathsf{S}_n$ such that $\mathbf{G}'\mathbf{P}\mathbf{H}^\top = \mathbf{0}$.
Output: Signature σ for the message m.
Compute the degree-D polynomial equations $\{f_1, \ldots, f_t\} \subset \mathbb{F}_{2^m}[X_1, \ldots, X_\ell]$ from pk and a solution $\mathbf{w} \in \mathbb{F}_2^\ell$ from sk;

$\mathbf{r} \xleftarrow{\$} \{0,1\}^\lambda$;
$\mu \leftarrow \mathsf{H}(\mathsf{pk}||\mathsf{m})$;
$\mathsf{iv} \leftarrow \mathsf{H}(\mathsf{sk}||\mu||\mathbf{r})$;
$(\mathsf{com}, \mathsf{decom}, \mathbf{u}_1, \mathbf{v}_1, \ldots, \mathbf{u}_\tau, \mathbf{v}_\tau) \leftarrow \mathsf{TC.Commit}(\mathbf{r}, \mathsf{iv}, \ell_{\mathsf{TC}})$;
for $z \in [\tau]$ **do**
  $\mathbf{t}_z \leftarrow \mathbf{w} - \mathbf{u}_{z\,[1..\ell]}$;
$\mathsf{ch}_1 \leftarrow \mathsf{H}(\mu||\mathsf{com}||\mathbf{t}_1||\ldots||\mathbf{t}_\tau||\mathsf{iv})$;
$\mathbf{\Gamma} \leftarrow \mathsf{ExpandGamma}(\mathsf{ch}_1)$;
for $z \in [\tau]$ **do**
  $\mathbf{u} \leftarrow \mathbf{u}_z$;
  $\mathbf{v} \leftarrow \mathbf{v}_z$;
  for $j \in [\rho]$ **do**
    for $i \in [D-1]$ **do**
      $\overline{u}_{j,i} \leftarrow \mathsf{ToField}((u_{\ell+(j\cdot(D-1)+i)\cdot r+1}, \ldots, u_{\ell+(j\cdot(D-1)+i+1)\cdot r}), r)$;
      $\overline{v}_{j,i} \leftarrow \mathsf{ToField}((v_{\ell+(j\cdot(D-1)+i)\cdot r+1}, \ldots, v_{\ell+(j\cdot(D-1)+i+1)\cdot r}), r)$;
  $\mathbf{a}_z \leftarrow \mathsf{NIZK.Prove}(\mathbf{w}, (f_1, \ldots, f_t), (v_j)_{j\in[\ell]}, (\overline{u}_{j,i}, \overline{v}_{j,i})_{j\in[\rho], i\in[D-1]}, \rho, \mathbf{\Gamma})$;
$\mathsf{ctr} = -1$;
repeat
  $\mathsf{ctr} = \mathsf{ctr} + 1$;
  $\mathsf{ch}_2 \leftarrow \mathsf{H}(\mathsf{ch}_1||\mathbf{a}_1||\ldots||\mathbf{a}_\tau||\mathsf{ctr})$;
  if $\mathsf{ch}_2[\lambda - w_{\mathsf{grind}} + 1, \lambda] \neq 0^{w_{\mathsf{grind}}}$ **then**
    continue;
  $I \leftarrow \mathsf{ComputeDelta}(\mathsf{ch}_2)$;
  $\mathsf{decom}_I \leftarrow \mathsf{BAVC.Open}(\mathsf{decom}, I, \mathsf{iv})$;
until $\mathsf{decom}_I \neq \perp$;
for $z \in [\tau]$ **do**
  parse $\mathbf{a}_z = ((a_{j,0}, \ldots, a_{j,D-1}))_{j\in[\rho]}$;
  $\widetilde{\mathbf{a}}_z \leftarrow ((a_{j,1}, \ldots, a_{j,D-1}))_{j\in[\rho]}$; // Remove the constant term of each polynomial
return $\sigma \leftarrow (\mathbf{t}_1, \ldots, \mathbf{t}_\tau, \widetilde{\mathbf{a}}_1, \ldots, \widetilde{\mathbf{a}}_\tau, \mathsf{decom}_I, \mathsf{ch}_2, \mathsf{iv}, \mathsf{ctr})$

4.4 Parameters and signature sizes

To choose the parameters for our signatures, we considered two main aspects. Firstly, as mentioned in Section 2, we choose instances of SEP that have, on average, only one solution. Recall that, for our model to be correct, we consider codes defined over fields of characteristic 2, i.e. with $q = 2^m$ for some $m \geq 1$. Then, thanks to Equation (1), to obtain instances of SEP with, on average, a unique solution we choose

$$m \approx \frac{1}{k'(1-R)} \cdot \log_2(n/e)\,.$$

Secondly, we considered the existing attacks for SEP, namely KMP [20], SBC [18] and BDP [21], and we computed parameter suites that achieved the desired security level against these attacks. In particular, we found secure parameters for the target security level $\lambda = 128$. In Table 2 we provide some parameter suites. We note that we focused on instances with m even to be able to embed the polynomial equations in $\mathbb{F}_{2^{128}}$ in the VOLEitH context.

Algorithm 8: Protocol $\mathsf{SETH}_{\mathsf{Verify-TC}}(\sigma, \mathsf{m}, \mathsf{pk}, \mathsf{pp})$

Input: Signature σ, signed message m, public key pk.
Output: Accept or reject.
Compute the degree-D polynomial equations $\{f_1, \ldots, f_t\} \subset \mathbb{F}_{2^m}[X_1, \ldots, X_\ell]$ from pk;
parse $\sigma = (\mathbf{t}_1, \ldots, \mathbf{t}_\tau, \mathbf{a}_1, \ldots, \mathbf{a}_\tau, \mathsf{decom}_I, \mathsf{ch}_2, \mathsf{iv}, \mathtt{ctr})$;
if $\mathsf{ch}_2[\lambda - w_{\mathsf{grind}} + 1, \lambda] \neq 0^{w_{\mathsf{grind}}}$ **then**
  reject;
$\mu \leftarrow \mathsf{H}(\mathsf{pk}||\mathsf{m})$;
$I \leftarrow (\Delta_1, \ldots, \Delta_\tau) = \mathsf{ComputeDelta}(\mathsf{ch}_2)$;
$(\overline{\mathsf{com}}, \mathbf{d}_1, \ldots, \mathbf{d}_\tau) \leftarrow \mathsf{TC.Reconstruct}(I, \mathsf{decom}_I, \mathsf{iv}, \ell + r \cdot \rho \cdot (D-1))$;
$\overline{\mathsf{ch}_1} \leftarrow \mathsf{H}(\mu||\overline{\mathsf{com}}||\mathbf{t}_1|| \ldots ||\mathbf{t}_\tau||\mathsf{iv})$;
$\overline{\mathbf{\Gamma}} \leftarrow \mathsf{ExpandGamma}(\overline{\mathsf{ch}_1})$;
for $z \in [\tau]$ **do**
  $\mathbf{d} \leftarrow \mathbf{d}_z$;
  for $j \in [\rho]$ **do**
    for $i \in [D-1]$ **do**
      $\overline{d}_{j,i} \leftarrow \mathsf{ToField}((d_{\ell+(j\cdot(D-1)+i)\cdot r+1}, \ldots, d_{\ell+(j\cdot(D-1)+i+1)\cdot r}), r)$;
  $(\overline{a}_{1,0}, \ldots, \overline{a}_{\rho,0}) \leftarrow \mathsf{NIZK.Verify}(\Delta_z, (f_1, \ldots, f_t), (d_j)_{j\in[\ell]}, (\overline{d}_{j,i})_{j\in[\rho], i\in[D-1]}, \mathbf{a}_z, \rho, \overline{\mathbf{\Gamma}})$;
  parse $\mathbf{a}_z = ((a_{j,1}, \ldots, a_{j,D-1}))_{j\in[\rho]}$;
  $\overline{\mathbf{a}}_z \leftarrow ((\overline{a}_{j,0}, a_{j,1}, \ldots, a_{j,D-1}))_{j\in[\rho]}$;
$\overline{\mathsf{ch}_2} \leftarrow \mathsf{H}(\overline{\mathsf{ch}_1}||\overline{\mathbf{a}}_1|| \ldots ||\overline{\mathbf{a}}_\tau||\mathtt{ctr})$;
Accept the signature if $\overline{\mathsf{ch}_2} = \mathsf{ch}_2$, otherwise reject.

Remark 6 (First vs. second modelling)**.** Overall, compared to the modelling from Section 3.1, the modelling from Section 3.2 increases the number of polynomial equations and the length of the representation of the elementary vectors: indeed, from Table 2 it can be seen that $\lceil \log_2(n) \rceil = 7 < 10$ for any of the proposed parameter sets. However, it significantly decreases the degree of the equations from $\lceil \log_2(n) \rceil = 7$ to 3. In particular, the latter significantly impacts the number of degree-1 polynomials that are needed to compute the masking polynomial in the PIOP.

Instead, the parameters τ and $N = 2^r$ for the PIOP are chosen according to the constraints described in Section 2.5 and differ for the two modellings described in Section 3: this is due to the dependence of the soundness error on the degree of the equations and changing this parameter will mostly influence the choice of w_{grind}. The protocol-related parameters are summarized in Table 1.

Remark 7. As in [3, 4], we must set τ, r and w_{grind} so that $\tau \cdot r + w_{\mathsf{grind}} > \lambda$ to achieve the correct soundness error. However, again as in [3, 4], $\tau \cdot r \leq \lambda$: in the VOLEitH version of SETH, this implies that we can perform computations in $\mathbb{F}_{2^\lambda}$ by embedding all the degree one polynomials and the last challenge Δ in $\mathbb{F}_{2^\lambda}$. As mentioned above, the polynomials can be naturally lifted to $\mathbb{F}_{2^\lambda}$ since we choose the extension degree m to divide λ. Moreover, the embedding of Δ in $\mathbb{F}_{2^\lambda}$ does not decrease the soundness of the protocol because the extra bits of security are still guaranteed by the grinding proof-of-work. Notice that, in this case, also the challenge $(\gamma_1, \ldots, \gamma_t)$ is sampled from $\mathbb{F}_{2^\lambda}^t$. With this strategy, the length of the PRG output for the VOLEitH case becomes

$$\hat{\ell}_{\mathsf{VOLE}} = \ell + (D-1) \cdot \lambda + \lambda + 16$$

since, now, the $(D-1)$ degree-1 polynomials used for masking must have coefficients in $\mathbb{F}_{2^\lambda}$. Signature sizes in Table 2 are computed using this length.

		VOLE				TC				
		τ	r	w_{grind}	T_{open}	ρ	τ	r	w_{grind}	T_{open}
Model 1	Short	11	11	10	102	10	12	13	8	136
	Fast	16	8	3	108	13	18	10	0	160
Model 2	Short	11	11	9	102	10	11	13	4	130
	Fast	16	8	2	108	13	16	10	0	145

Table 1: Parameters related to the security of the underlying PIOP and the VOLEitH and TCitH constructions.

In Table 2, we report the signature sizes obtained by applying both the VOLEitH and the TCitH frameworks to the PIOP protocol with the two SEP modellings introduced in Section 3. It can be seen that, in general, the VOLEitH framework yields shorter signatures compared to the TCitH one. Moreover, it is also evident that the modelling from Section 3.2 yields shorter signatures compared to the one from Section 3.1. For these reasons, in the following section we benchmark SETH using VOLEitH and the second SEP modelling.

Instance				Attacks' Cost					Model 1		Model 2	
m	n	k'	k	KMP	SBC	BDP	pk		VOLE	TC	VOLE	TC
4	74	3	36	209.62	204.34	130.31	122.5	Short	3986.62	4547.5	3587.75	3845.38
								Fast	5131.38	6099.0	4556.25	4872.0
2	95	8	38	277.51	225.57	128.56	190.0	Short	4188.75	4768.0	3876.5	4134.12
								Fast	5425.38	6429.75	4976.25	5292.0
2	86	9	42	241.35	203.26	131.65	189.25	Short	4102.12	4673.5	3752.75	4010.38
								Fast	5299.38	6288.0	4796.25	5112.0
2	78	10	43	207.95	179.72	130.77	186.0	Short	4025.12	4589.5	3642.75	3900.38
								Fast	5187.38	6162.0	4636.25	4952.0
2	76	11	44	200.74	172.64	130.63	194.75	Short	4005.88	4568.5	3615.25	3872.88
								Fast	5159.38	6130.5	4596.25	4912.0

Table 2: Parameters, attacks' costs and signature sizes for the signature schemes obtained from the two modellings of SEP in the VOLEitH and TCitH frameworks. The sizes of the public key and the signatures are given in bytes.

4.5 Benchmarking and implementation

We derived an AVX2-optimized implementation of SETH based on the implementation of FAEST v2.0.2[6], available at https://github.com/secomms/SETH. The implementation was benchmarked against other signature schemes based on similar constructions, and compared in Figure 1. All schemes were benchmarked on an AMD Ryzen 7 PRO 8840U CPU clocked at 3.3 GHz with Precision Boost disabled. Table 3 reports the performance of SETH.

Instance	KeyGen		Sign		Verify		pk	SigSize	pk + Sig
	ms	MCycles	ms	MCycles	ms	MCycles	kB	kB	kB
SETH-short-4-74	0.03	0.11	5.86	19.30	2.69	8.85	0.12	3.59	3.72
SETH-short-2-95	0.08	0.25	6.23	20.52	2.84	9.35	0.19	3.88	4.07
SETH-short-2-86	0.06	0.21	6.09	20.07	2.70	8.88	0.19	3.76	3.95
SETH-short-2-78	0.05	0.18	5.81	19.14	2.65	8.73	0.19	3.65	3.83
SETH-short-2-76	0.05	0.18	5.86	19.30	2.72	8.96	0.20	3.62	3.81
SETH-fast-4-74	0.03	0.11	0.86	2.84	0.71	2.32	0.12	4.56	4.69
SETH-fast-2-95	0.08	0.25	0.92	3.02	0.71	2.35	0.19	4.98	5.17
SETH-fast-2-86	0.07	0.21	0.83	2.72	0.65	2.15	0.19	4.80	4.99
SETH-fast-2-78	0.06	0.18	0.78	2.56	0.63	2.07	0.19	4.64	4.83
SETH-fast-2-76	0.05	0.18	0.80	2.62	0.66	2.16	0.20	4.60	4.79

Table 3: Benchmarks for SETH instances at 128-bit security level. Numbers were measured on an AMD Ryzen 7 PRO 8840U CPU clocked at 3.3 GHz with Precision Boost disabled, as average of 1000 runs.

5 Conclusions

In this paper we introduced SETH, a family of post-quantum signature schemes based on the Subcode Equivalence Problem (SEP). We proposed two polynomial modellings of SEP over fields of characteristic 2. The first represents a permutation matrix through the binary encoding of the position of the non-zero entry of each row, and yields equations of degree $D = \lceil \log_2 n \rceil$. The second, adapted from the encodings of elementary vectors by Chou, uses more variables and equations but lowers the degree to $D = 3$, which reduces the number of masking polynomials required by the PIOP. Relying on the PIOP formalism, we derived both a TCitH and a VOLEitH signature from these modellings.

The VOLEitH variant reaches sizes between about 3.6 and 4.2 kB for the small variant and between about 4.5 and 5.4 kB for the fast variant, while the TCitH variant ranges between about 3.8 and 5 kB for the small variant and between about 4.9 and 6.7 kB for the fast variant, with public keys of at most 195 bytes. These sizes are very close to those of PERK and up to 1 kB shorter than the signature based on the Matrix Subcode Equivalence Problem, and about 10 times smaller than the VOLEitH signature based on the Linear Code Equivalence Problem.[7] Compared with LESS,

[6] Available at https://github.com/faest-sign/faest-arch-opt.
[7] Unfortunately, these two protocols lack an implementation, thus we are not able to compare the timings.

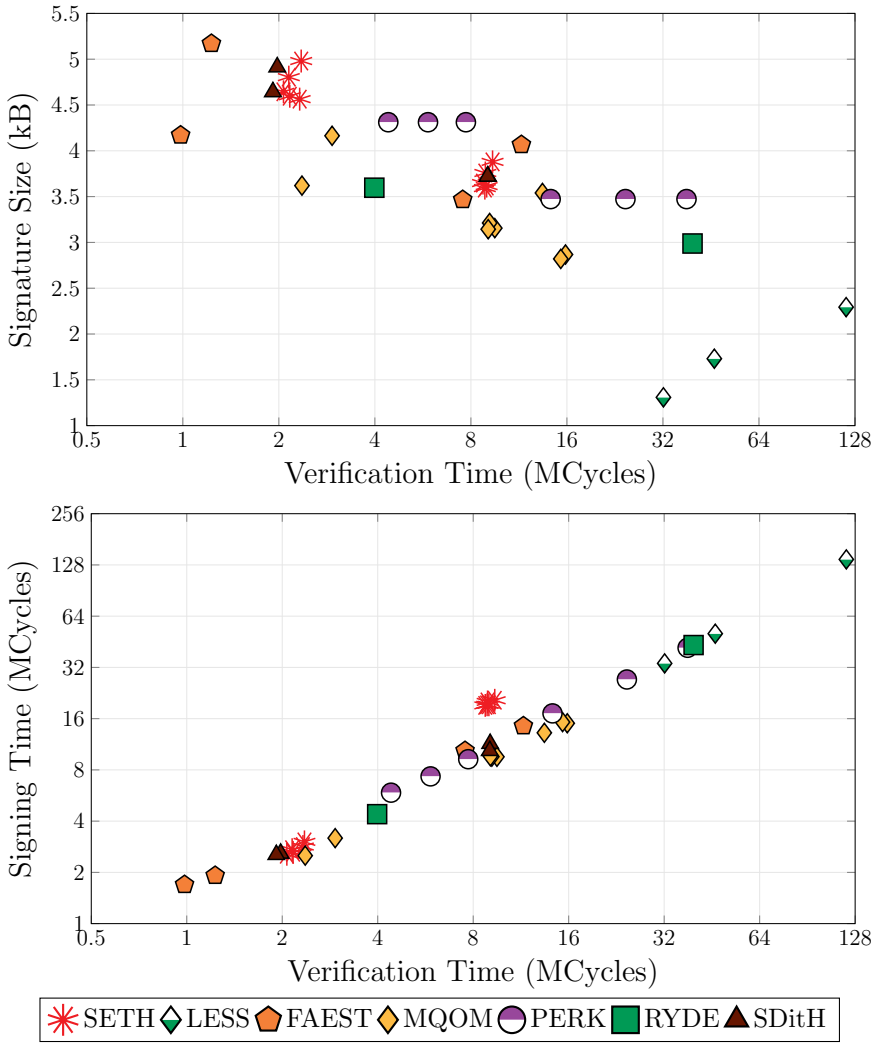

Figure 1: Overview of the performance for signature schemes based on TCitH and VOLEitH frameworks and LESS. Benchmarks were performed on an AMD Ryzen 7 PRO 8840U CPU clocked at 3.3 GHz with Precision Boost disabled.

our signatures are larger, but our public keys are around 70 times smaller, and the algorithms are significantly faster.

Compared with the other in-the-Head signatures, SETH lies in the same range of signature size and verification time. The fast instances are among the fastest schemes, at the price of being among those with the largest signatures. On the other hand, the short instances yield more compact signatures, at the price of a signing time that is among the slowest (with a slightly better verification time).

As for future work directions, given the performance of SETH, it would be interesting to study advanced signature schemes based on SETH such as (threshold) ring signatures, perhaps using standard constructions for in-the-Head schemes [32, 33].

Acknowledgements

This work is partially supported by the Italian Ministry of University and Research (MUR) under the Italian Fund for Applied Science (FISA 2022), Call for tender No. 1405 published on 13-09-2022 - project title "Quantum-safe cryptographic tools for the protection of national data and information technology assets" (QSAFEIT) - No. FISA 2022-00618 (CUP I33C24000520001), Grant Assignment Decree no. 15461 adopted on 02.08.2024, by the QUBIP project, funded by the European Union under the Horizon Europe framework programme [grant agreement no. 101119746] and by the University of St.Gallen, Great Mind Fellowship, Project 103 I 638 for "Code-Based Cryptography for Post-Quantum Security (CoB-PQS)".

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 [8] and later used in the first version of FAEST ([3]).

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 $N = 2^\delta$. The random vector commitment VC is defined in Algorithm 9. From Algorithm 9, 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 [8, Section 3.1].

A.2 All-but-τ Vector Commitment Scheme

As mentioned in Section 2.5.3, 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

Algorithm 9: All-but-one vector commitment ([34, Section 5.1, Figure 5.1])

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

$\mathsf{VC.Open}(\mathsf{decom}, j^* \in [0, N-1])$:
Write $j^* = \sum_{i=0}^{\delta-1} 2^i b_i$;
Parse $\mathsf{decom} = (((k_j^i)_{j \in [0, 2^i - 1]})_{i \in [\delta]}, (\overline{\mathsf{com}}_j)_{j \in [0, N-1]})$;
$a \leftarrow 0$;
for $i \in [\delta]$ **do**
  $\mathsf{cop}[i] \leftarrow k^i_{2a + \overline{b_{\delta - i}}}$;
  $a \leftarrow 2a + b_{\delta - i}$;
$\mathsf{pdecom} \leftarrow (\mathsf{cop}, \overline{\mathsf{com}}_{j^*})$;
return pdecom

$\mathsf{VC.Reconstruct}(\mathsf{pdecom}, j^* = \sum_{i=0}^{\delta-1} 2^i b_i)$:
Parse $\mathsf{pdecom} = (\mathsf{cop}, \overline{\mathsf{com}}_{j^*})$;
$a \leftarrow 0$, $k_0^0 \leftarrow \perp$;
for $i \in [\delta]$ **do**
  $k^i_{2a + \overline{b_{\delta - i}}} \leftarrow \mathsf{cop}[i]$;
  $k^i_{2a + b_{\delta - i}} \leftarrow \perp$;
  for $j \in [0, 2^{i-1} - 1] \backslash \{a\}$ **do**
    $(k_{2j}^i, k_{2j+1}^i) \leftarrow \mathsf{PRG}(k_j^{i-1})$;
  $a \leftarrow 2a + b_{\delta - i}$;
for $j \in [0, N-1] \backslash \{j^*\}$ **do**
  $(\mathsf{seed}_j, \overline{\mathsf{com}}_j) \leftarrow \mathsf{H}_1(k_j^\delta)$;
$h \leftarrow \mathsf{H}_0(\overline{\mathsf{com}}_0, ..., \overline{\mathsf{com}}_{N-1})$;
return $(h, (\mathsf{seed}_j)_{j \in [0, N-1] \backslash \{j^*\}})$

$\mathsf{VC.Verify}(\mathsf{com}, \mathsf{pdecom}, j^* = \sum_{i=0}^{\delta-1} 2^i b_i)$:
$(\overline{\mathsf{com}}, (\mathsf{seed}_j)_{j \in [0, N-1] \backslash \{j^*\}}) \leftarrow \mathsf{VC.Reconstruct}(\mathsf{pdecom}, j^*)$;
if $\mathsf{com} = \overline{\mathsf{com}}$ **then**
  return 1
else
  return 0

by employing a *batch all-but-one vector commitment scheme* (BAVC). Introduced in [30], a batch vector commitment scheme is defined as follows.

Definition 8 (BAVC [30, 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 of them have access to a random oracle and obtain the security parameter 1^λ as well as $\tau, N_0, ..., N_{\tau-1}$ as input:

- $\mathsf{Commit}() \rightarrow (\mathsf{com}, \mathsf{decom}, (m_0^{(\alpha)}, ..., m_{N_\alpha - 1}^{(\alpha)})_{\alpha \in [0, \tau-1]})$: Output a commitment com with opening information decom for messages $(m_0^{(\alpha)}, ..., m_{N_\alpha-1}^{(\alpha)})_{\alpha \in [0,\tau-1]} \in \mathcal{M}^{N_0+...+N_{\tau-1}}$;
- $\mathsf{Open}(\mathsf{decom}, I) \rightarrow \mathsf{decom}_I \vee \perp$: On input an opening decom and the index vector $I \in [0, N_0 - 1] \times \cdots \times [0, N_{\tau-1} - 1]$, output $\perp$ or an opening decom_I for I;
- $\mathsf{Verify}(\mathsf{com}, \mathsf{decom}_I, I) \rightarrow ((m_j^{(\alpha)})_{j \in [0, N_\alpha - 1] \setminus \{i^{(\alpha)}\}})_{\alpha \in [0,\tau-1]} \vee \perp$: Given a commitment com, an opening decom_I for an index vector I, and the index vector I, either output all messages $(m_j^{(\alpha)})_{j \in [0, N_\alpha - 1] \setminus \{i^{(\alpha)}\}}$ (accept the opening) or $\perp$ (reject the opening).

As previously mentioned, we can express τ single VCs as one BAVC. The idea is to generate a single GGM tree with $\sum_{\alpha=0}^{\tau-1} N_\alpha$ leaves and open all but 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) $\mathsf{PRG} : \{0,1\}^\lambda \times \{0,1\}^{128} \rightarrow \{0,1\}^{2\lambda}$;
- a collision-resistant hash function (CRHF) $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^{2\lambda}$;
- a PRG and CRHF $\mathsf{G} : \{0,1\}^\lambda \times \{0,1\}^{128} \rightarrow \{0,1\}^\lambda \times \{0,1\}^{2\lambda}$;
- the number of vectors τ;
- the lengths of the vectors $N_0, ..., N_{\tau-1}$;
- the opening threshold T_{open};
- $\pi : [L-1, 2L-2] \rightarrow \{(\alpha, i)\}_{0 \leq i \leq N_\alpha - 1}$ a bijective mapping from the leaves of the GGM tree to the positions in the vector commitments, where $L = \sum_{\alpha=0}^{\tau-1} N_\alpha$ is the number of leaves.

The BAVC is defined in Algorithm 10, where $\mathsf{BAVC.Reconstruct}$ recomputes the commitment and the opened messages, so that Verify accepts if and only if the recomputed commitment equals com. The definition of the algorithms is adapted from [30] and [3]. From Algorithm 10 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 [30, Section 3], while the respective proofs can be found in [30, Lemma 6, Lemma 7].

Appendix B Generating VOLE Correlations

We report here the algorithms used to generate and reconstruct VOLE correlations. We focus on the generation procedure exploiting a BAVC, as done in the second version of FAEST [3], instead of τ all-but-one vector commitment schemes, as in the first version of FAEST [3]. We adapt the algorithms defined in [29, Section 5.5] to our framework, where we need to generate $\tau \geq (\lambda + \log_2(D))/(r \log_2(q))$ VOLE correlations over $\mathbb{F}_{q^r}$ and adjust them to obtain a single correlation over $\mathbb{F}_{q^{r\tau}}$. Hence, hereafter $N_0 = \cdots = N_{\tau-1} = N = q^r$, and the single VOLE correlations will be computed

Algorithm 10: Batch all-but-one vector commitment

$\mathsf{BAVC.Commit}(r \in \{0,1\}^\lambda, \mathsf{iv} \in \{0,1\}^{128})$:
$k_0 \leftarrow r$;
for $i \in [0, L-2]$ **do**
  $(k_{2i+1}, k_{2i+2}) \leftarrow \mathsf{PRG}(k_i, \mathsf{iv})$;
$\{\mathsf{seed}_0^{(\alpha)}, ..., \mathsf{seed}_{N_\alpha-1}^{(\alpha)}\}_{\alpha \in [0,\tau-1]} \xleftarrow{\pi} \{k_{L-1}, ..., k_{2L-2}\}$;
for $\alpha \in [0, \tau-1]$ **do**
  for $i \in [0, N_\alpha - 1]$ **do**
    $(m_i^{(\alpha)}, \mathsf{com}_i^{(\alpha)}) \leftarrow \mathsf{G}(\mathsf{seed}_i^{(\alpha)}, \mathsf{iv})$;
for $\alpha \in [0, \tau-1]$ **do**
  $h^{(\alpha)} \leftarrow \mathsf{H}(\mathsf{com}_0^{(\alpha)}, ..., \mathsf{com}_{N_\alpha-1}^{(\alpha)})$;
$\mathsf{com} = h \leftarrow \mathsf{H}(h^{(0)}, ..., h^{(\tau-1)})$;
$\mathsf{decom} \leftarrow k_0$;
return $\left(\mathsf{com}, \mathsf{decom}, (m_i^{(\alpha)})_{i \in [0,N_\alpha-1], \alpha \in [0,\tau-1]}\right)$

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

$\mathsf{BAVC.Reconstruct}(\mathsf{decom}_I, I, \mathsf{iv})$:
Parse $\mathsf{decom}_I = ((\mathsf{com}_{i^{(\alpha)}}^{(\alpha)})_{\alpha \in [0,\tau-1]}, S)$ and $I = (i^{(0)}, ..., i^{(\tau-1)})$;
Recompute $\mathsf{seed}_i^{(\alpha)}$ from decom_I for $\alpha \in [0, \tau-1]$ and $i \neq i^{(\alpha)}$ using the internal nodes in S;
Compute $(m_i^{(\alpha)}, \mathsf{com}_i^{(\alpha)}) \leftarrow \mathsf{G}(\mathsf{seed}_i^{(\alpha)}, \mathsf{iv})$ for $\alpha \in [0, \tau-1]$ and $i \neq i^{(\alpha)}$;
for $\alpha \in [0, \tau-1]$ **do**
  $h^{(\alpha)} \leftarrow \mathsf{H}(\mathsf{com}_0^{(\alpha)}, ..., \mathsf{com}_{N_\alpha-1}^{(\alpha)})$;
$h \leftarrow \mathsf{H}(h^{(0)}, ..., h^{(\tau-1)})$;
return $\left(h, (m_i^{(\alpha)})_{i \in [0,N_\alpha-1] \setminus \{i^{(\alpha)}\}, \alpha \in [0,\tau-1]}\right)$

over $\mathbb{F}_{q^r}$ as in Section 2.5. As a last bit of notation, in what follows we let PRG_q be a pseudorandom generator that, given as input a seed $\mathsf{seed} \in \{0,1\}^*$, an initialization vector $\mathsf{iv} \in \{0,1\}^{128}$ and a positive integer $\hat{\ell}_{\mathsf{VOLE}} \in \mathbb{N}$, returns a vector of the specified length $\hat{\ell}_{\mathsf{VOLE}}$ over $\mathbb{F}_q$. The procedures are defined in Algorithm 11.

Appendix C VOLE Consistency Check

In this section, we provide more details about the consistency check for VOLE correlations. Briefly, to ensure that the corrections $\mathbf{e}_1, \ldots, \mathbf{e}_{\tau-1}$ are consistent with the committed VOLE correlation, the verifier asks the prover to compute a random linear hash function. This procedure is generic and does not depend on the specific problem or the underlying PoK. Hence, we employ the same algorithms for these checks used in FAEST [3] and PERK [9]. In particular, notice that PERK also uses the same techniques as in FAEST, but it generalizes it to the case of polynomial constraints of

Algorithm 11: Conversion from seeds to VOLE correlations

$\mathsf{P.ConvertToVOLE}(\mathsf{seed}_0, \ldots, \mathsf{seed}_{N-1}, \mathsf{iv}, L)$:
for $j \in [0, N-1]$ **do**
  $\mathbf{s}^{(j)} \leftarrow \mathsf{PRG}_q(\mathsf{seed}_j, \mathsf{iv}, L)$;
$\mathbf{u} \leftarrow \sum_{j=0}^{N-1} \mathbf{s}^{(j)}$;
$\mathbf{v} \leftarrow -\sum_{j=0}^{N-1} j \cdot \mathbf{s}^{(j)}$;
return $(\mathbf{u}, \mathbf{v})$

$\mathsf{V.ConvertToVOLE}(\Delta, (\mathsf{seed}_j)_{j \in [0,N-1] \setminus \{\Delta\}}, \mathsf{iv}, L)$:
for $j \in [0, N-1] \setminus \{\Delta\}$ **do**
  $\mathbf{s}^{(j)} \leftarrow \mathsf{PRG}_q(\mathsf{seed}_j, \mathsf{iv}, L)$;
$\mathbf{d} \leftarrow \sum_{j \neq \Delta} (\Delta - j) \cdot \mathbf{s}^{(j)}$;
return d

degree ≥ 2.

As already mentioned, the consistency check employs linear universal hash functions.

Definition 9 (Linear universal hash functions)**.** A family of linear hash functions is a family of matrices $\mathcal{H} \subseteq \mathbb{F}_q^{r \times n}$. The family is *$\varepsilon$-almost universal* if for any non-zero $\mathbf{x} \in \mathbb{F}_q^n$

$$\Pr_{\mathbf{A} \xleftarrow{\$} \mathcal{H}}[\mathbf{A}\mathbf{x} = \mathbf{0}] \leq \varepsilon.$$

The family is *ε-almost uniform* if for any non-zero $\mathbf{x} \in \mathbb{F}_q^n$ and for any $\mathbf{v} \in \mathbb{F}_q^r$

$$\Pr_{\mathbf{A} \xleftarrow{\$} \mathcal{H}}[\mathbf{A}\mathbf{x} = \mathbf{v}] \leq \varepsilon.$$

The linear hash functions used in this context must also satisfy the following additional property.

Definition 10. A matrix $\mathbf{A} \in \mathbb{F}_q^{r \times (n+h)}$ is *$\mathbb{F}_q^n$-hiding* if the distribution of $\mathbf{A}\mathbf{v}$ is independent of $\mathbf{v}_{[1..n]}$ when $\mathbf{v}_{[n+1,\ldots,n+h]} \xleftarrow{\$} \mathbb{F}_q^h$. A hash family $\mathcal{H} \subseteq \mathbb{F}_q^{r \times (n+h)}$ is $\mathbb{F}_q^n$-hiding if every $\mathbf{A} \in \mathcal{H}$ is $\mathbb{F}_q^n$-hiding.

The construction consists of choosing a uniform hash family and transforming it into a universal hash family.

Proposition 3. *Let $\mathcal{H} \subseteq \mathbb{F}_q^{r \times n}$ be an ε-almost uniform hash family. Let $\mathcal{H}' \subseteq \mathbb{F}_q^{r \times (n+r)}$ be the family $\{[\mathbf{H}\ \mathbf{I}_r] \mid \mathbf{H} \in \mathcal{H}\}$. Then, it holds that $\mathcal{H}'$ is ε-almost universal and $\mathbb{F}_q^n$-hiding.*

The proof of this proposition can be found at [29, Proposition 4.6].

Algorithm 12: VOLE-in-the-Head and TC-in-the-Head functions

$\mathsf{VOLE.Commit}(\mathbf{r}, \mathsf{iv}, \hat{\ell}_{\mathsf{VOLE}})$:
$(\mathsf{com}, \mathsf{decom}, (\mathsf{seed}_0^{(i)}, ..., \mathsf{seed}_{N_i-1}^{(i)})_{i\in[\tau]}) \leftarrow \mathsf{BAVC.Commit}(\mathbf{r}, \mathsf{iv})$;
for $i \in [\tau]$ **do**
  $(\mathbf{u}_i, \mathbf{v}_i) \leftarrow \mathsf{P.ConvertToVOLE}(\mathsf{seed}_0^{(i)}, ..., \mathsf{seed}_{N_i-1}^{(i)}, \mathsf{iv}, \hat{\ell}_{\mathsf{VOLE}})$;
$\mathbf{V} \leftarrow [\mathbf{v}_1 \mid \cdots \mid \mathbf{v}_\tau]$;
$\mathbf{u} \leftarrow \mathbf{u}_1$;
for $i \in [2, \tau]$ **do**
  $\mathbf{e}_i \leftarrow \mathbf{u}_i - \mathbf{u}$;
return $(\mathsf{com}, \mathsf{decom}, \mathbf{e}_2, ..., \mathbf{e}_\tau, \mathbf{u}, \mathbf{V})$

$\mathsf{VOLE.Reconstruct}(I, \mathsf{decom}_I, \mathbf{e}_2, ..., \mathbf{e}_\tau, \mathsf{iv}, \hat{\ell}_{\mathsf{VOLE}})$:
if $I = \perp$ **then return** $\perp$;
$\mathsf{out} = \left(\mathsf{com}, (\mathsf{seed}_j^{(i)})_{j\in[0,N_i-1]\setminus\{\Delta_i\}, i\in[\tau]}\right) \leftarrow \mathsf{BAVC.Reconstruct}(\mathsf{decom}_I, I, \mathsf{iv})$;
if $\mathsf{out} = \perp$ **then return** $\perp$;
$\mathbf{e}_1 \leftarrow \mathbf{0}$;
for $i \in [\tau]$ **do**
  $\mathbf{d}'_i \leftarrow \mathsf{V.ConvertToVOLE}(\Delta_i, (\mathsf{seed}_j^{(i)})_{j\in[0,N_i-1]\setminus\{\Delta_i\}}, \mathsf{iv}, \hat{\ell}_{\mathsf{VOLE}})$;
  $\mathbf{d}_i \leftarrow \mathbf{d}'_i - \mathbf{e}_i \cdot \Delta_i$;
$\mathbf{D} = [\mathbf{d}_1 \mid \cdots \mid \mathbf{d}_\tau]$;
return $(\mathsf{com}, \mathbf{D})$

$\mathsf{TC.Commit}(\mathbf{r}, \mathsf{iv}, \ell_{\mathsf{TC}})$:
$(\mathsf{com}, \mathsf{decom}, (\mathsf{seed}_0^{(i)}, ..., \mathsf{seed}_{N_i-1}^{(i)})_{i\in[\tau]}) \leftarrow \mathsf{BAVC.Commit}(\mathbf{r}, \mathsf{iv})$;
for $i \in [\tau]$ **do**
  $(\mathbf{u}_i, \mathbf{v}_i) \leftarrow \mathsf{P.ConvertToVOLE}(\mathsf{seed}_0^{(i)}, ..., \mathsf{seed}_{N_i-1}^{(i)}, \mathsf{iv}, \ell_{\mathsf{TC}})$;
return $(\mathsf{com}, \mathsf{decom}, \mathbf{u}_1, \mathbf{v}_1, ..., \mathbf{u}_\tau, \mathbf{v}_\tau)$

$\mathsf{TC.Reconstruct}(I, \mathsf{decom}_I, \mathsf{iv}, \ell_{\mathsf{TC}})$:
if $I = \perp$ **then return** $\perp$;
$\mathsf{out} = \left(\mathsf{com}, (\mathsf{seed}_j^{(i)})_{j\in[0,N_i-1]\setminus\{\Delta_i\}, i\in[\tau]}\right) \leftarrow \mathsf{BAVC.Reconstruct}(\mathsf{decom}_I, I, \mathsf{iv})$;
if $\mathsf{out} = \perp$ **then return** $\perp$;
for $i \in [\tau]$ **do**
  $\mathbf{d}_i \leftarrow \mathsf{V.ConvertToVOLE}(\Delta_i, (\mathsf{seed}_j^{(i)})_{j\in[0,N_i-1]\setminus\{\Delta_i\}}, \mathsf{iv}, \ell_{\mathsf{TC}})$;
return $(\mathsf{com}, \mathbf{d}_1, \ldots, \mathbf{d}_\tau)$

C.1 VOLE universal hash

The construction of the VOLE universal hash family uses two constructions of linear universal hash families [35, 36]. The first one is the matrix hash family $\mathcal{H} = \mathbb{F}_q^{r\times n}$, which is q^{-r}-almost uniform. The second one, instead, is a polynomial-based hash where an input of length n is parsed as the coefficients of a polynomial of degree at most $n-1$, and sampling a hash function from the family consists of evaluating the polynomial at a randomly chosen point in $\mathbb{F}_q$. Since the polynomial has at most $n-1$ roots over $\mathbb{F}_q$, the family is $(n-1)/q$-almost universal.

The universal hash family for the VOLE consistency check is linear over $\mathbb{F}_2$ and is evaluated on inputs in $\mathbb{F}_2^{\hat{\ell}}$, with $\hat{\ell} = \ell + D\cdot\lambda + B$, where $B = 16$ is a parameter chosen for security. The specific hash function is sampled from a seed seed which is parsed as $(r_0, r_1, r_2, r_3, s, t) \in \mathbb{F}_{2^\lambda}^5 \times \mathbb{F}_{2^{64}}$. The input $\mathbf{x} \in \mathbb{F}_2^{\hat{\ell}}$ to the hash function is split into $(\mathbf{x}_0, \mathbf{x}_1)$, with $\mathbf{x}_0 \in \{0,1\}^{\ell+(D-1)\lambda}$ and $\mathbf{x}_1 \in \{0,1\}^{\lambda+B}$. Then, $\mathbf{x}_0$ is zero-padded to length $\ell' = \lambda \cdot \lceil(\ell + (D-1)\lambda)/\lambda\rceil$ and parsed first as a vector $\hat{\mathbf{y}}$ of elements in $\mathbb{F}_{2^\lambda}$

Algorithm 13: Hash function for VOLE consistency check

$\mathsf{VOLEHash}(\mathsf{seed}, (\mathbf{x}_0, \mathbf{x}_1) \in \{0,1\}^{\ell+(D-1)\lambda} \times \{0,1\}^{\lambda+B})$:
parse $\mathsf{seed} = (\mathbf{r}_0||\mathbf{r}_1||\mathbf{r}_2||\mathbf{r}_3||\mathbf{s}||\mathbf{t}) \in \{0,1\}^{5\lambda+64}$;
for $i \in [0,3]$ **do**
  $r_i = \mathsf{ToField}(\mathbf{r}_i, \lambda)$;
$s = \mathsf{ToField}(\mathbf{s}, \lambda)$;
$t = \mathsf{ToField}(\mathbf{t}, 64)$;
$\ell' = \lambda \cdot \lceil(\ell + (D-1)\cdot\lambda)/\lambda\rceil$;
$\mathbf{x}_0 = \mathbf{x}_0||0^{\ell'-(\ell+(D-1)\cdot\lambda)}$;
$\hat{\mathbf{y}} = \mathsf{ToField}(\mathbf{x}_0, \lambda)$;
$\overline{\mathbf{y}} = \mathsf{ToField}(\mathbf{x}_0, 64)$;
$h_0 = \sum_{i=0}^{\ell'/\lambda-1} s^{\ell'/\lambda-1-i} \cdot \hat{y}_i$ (in $\mathbb{F}_{2^\lambda}$);
$h_1 = \sum_{i=0}^{\ell'/64-1} t^{\ell'/64-1-i} \cdot \overline{y}_i$ (in $\mathbb{F}_{2^{64}}$);
$h_1' = \mathsf{ToField}(\mathsf{ToBits}(h_1)||0^{\lambda-64}, \lambda)$;
$(h_2, h_3) = (r_0 h_0 + r_1 h_1', r_2 h_0 + r_3 h_1')$;
$\mathbf{h} = (\mathsf{ToBits}(h_2)||\mathsf{ToBits}(h_3)[0..B)) \oplus \mathbf{x}_1$;
return h

and then as a vector $\overline{\mathbf{y}}$ of elements in $\mathbb{F}_{2^{64}}$. Then, compute

$$h_0 = \hat{y}_0 \cdot s^{\ell'/\lambda-1} + \hat{y}_1 \cdot s^{\ell'/\lambda-2} + \ldots + \hat{y}_{\ell'/\lambda-2} \cdot s + \hat{y}_{\ell'/\lambda-1} \text{ in } \mathbb{F}_{2^\lambda}$$
$$h_1 = \overline{y}_0 \cdot t^{\ell'/64-1} + \overline{y}_1 \cdot t^{\ell'/64-2} + \ldots + \overline{y}_{\ell'/64-2} \cdot t + \overline{y}_{\ell'/64-1} \text{ in } \mathbb{F}_{2^{64}}$$

Viewing h_1 as an element of $\mathbb{F}_{2^\lambda}$ by padding zeroes, the hash is then defined by computing in $\mathbb{F}_{2^\lambda}$

$$\begin{bmatrix} h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} r_0 & r_1 \\ r_2 & r_3 \end{bmatrix} \cdot \begin{bmatrix} h_0 \\ h_1 \end{bmatrix}$$

Finally, the output is given by the XOR between $\mathbf{x}_1$ and the first $\lambda + B$ bits of the concatenation of the field elements h_2 and h_3. The security of the construction is argued in [29, Lemma 4.10] (adapted to larger degrees in [9, Lemma 4.1]).

The algorithm computing the hash function for the VOLE consistency check is found in Algorithm 13.

Appendix D Additional functions

In this section, we simply report the additional functions for type conversion between field elements and bit strings.

Algorithm 14: Additional algorithms

ToField($\mathbf{x}, k$):
let $\mathbf{x} = (x_0, \ldots, x_{n-1})$;
let $\alpha_k \in \mathbb{F}_{2^k}$; // The primitive element of $\mathbb{F}_{2^k}$
if $\mathbf{x} \in \{0,1\}^k$ **then**
  return $x = \sum_{i=0}^{k-1} x_i \cdot \alpha_k^i$;
else if $\mathbf{x} \in \{0,1\}^{nk}$ **then**
  new $\mathbf{y} \in \mathbb{F}_{2^k}^n$;
  for $i \in [0, n-1]$ **do**
    $y_i = \sum_{j=0}^{k-1} x_{k \cdot i + j} \cdot \alpha_k^j$;
  return $\mathbf{y} = (y_0, \ldots, y_{n-1})$;
else
  return $\perp$;

ToBits($\mathbf{y}; k, n$):
let $\mathbf{y} = (y_0, \ldots, y_{n-1}) \in \mathbb{F}_{2^k}^n$;
for $i \in [0, n-1]$ **do**
  parse $y_i = x_0 + x_1 \cdot \alpha_k + \ldots + x_{k-1} \cdot \alpha_k^{k-1}$ for $x_j \in \mathbb{F}_2$;
  $\mathbf{x}_i = (x_0, \ldots, x_{k-1}) \in \{0,1\}^k$;
return $\mathbf{x} = (\mathbf{x}_0 || \cdots || \mathbf{x}_{n-1})$

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] Benadjila, R., Bouillaguet, C., Feneuil, T., Rivain, M.: MQOM — MQ on my Mind. Technical report, National Institute of Standards and Technology (2026). available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-3-additional-signatures

[3] Baum, C., Braun, L., Beullens, W., Delpech de Saint Guilhem, C., Klooß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: FAEST. Technical report, National Institute of Standards and Technology (2026). available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-3-additional-signatures

[4] 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 (2026). available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-3-additional-signatures

[5] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing. STOC '07, pp. 21–30. Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1250790.

1250794 . https://doi.org/10.1145/1250790.1250794

[6] Feneuil, T., Rivain, M.: Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments. J. Cryptol. **38**(3) (2025) https://doi.org/10.1007/s00145-025-09543-8

[7] Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)

[8] 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)

[9] Aaraj, N., Bettaieb, S., Bidoux, L., Budroni, A., Dyseryn, V., Esser, A., Feneuil, T., Gaborit, P., Kulkarni, M., Mateu, V., Palumbi, M., Perin, L., Rivain, M., Tillich, J., Xagawa, K.: PERK. Technical report, National Institute of Standards and Technology (2024). available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures

[10] Berger, T.P., Gueye, C.T., Klamti, J.B.: A NP-complete problem in coding theory with application to code based cryptography. In: El Hajji, S., Nitaj, A., Souidi, E.M. (eds.) Codes, Cryptology and Information Security, pp. 230–237. Springer, Cham (2017)

[11] Girault, M.: A (non-practical) three-pass identification protocol using coding theory. In: Seberry, J., Pieprzyk, J. (eds.) Advances in Cryptology — AUSCRYPT '90, pp. 265–272. Springer, Berlin, Heidelberg (1990)

[12] Bardet, M., Brion, C., Gaborit, P., Haiech, M., Neveu, R.: The matrix subcode equivalence problem and its application to signature with MPC-in-the-head. Designs, Codes and Cryptography **94**(76) (2026) https://doi.org/10.1007/s10623-026-01836-8

[13] 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

[14] Battagliola, M., Mattiuz, L., Meneghetti, A.: VOLE-in-the-head signatures based on the linear code equivalence problem. Cryptography and Communications (2026) https://doi.org/10.1007/s12095-026-00869-z

[15] Baldi, M., Barenghi, A., Beckwith, L., Biasse, J., Chou, T., Esser, A., Gaj, K., Karl, P., Mohajerani, K., Pelosi, G., Persichetti, E., Saarinen, M.O., Santini, P., Wallace, R., Zweydinger, F.: LESS — Linear Equivalence Signature Scheme. Technical report, National Institute of Standards and Technology (2024). available at https://csrc.nist.gov/Projects/pqc-dig-sig/

round-2-additional-signatures

[16] Petrank, E., Roth, R.M.: Is code equivalence easy to decide? IEEE Transactions on Information Theory **43**(5), 1602–1604 (1997) https://doi.org/10.1109/18.623157

[17] 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)

[18] Santini, P., Baldi, M., Chiaraluce, F.: Computational hardness of the permuted kernel and subcode equivalence problems. IEEE Transactions on Information Theory **70**(3), 2254–2270 (2024) https://doi.org/10.1109/TIT.2023.3323068

[19] Sanna, C.: On the number of solutions to a random instance of the permuted kernel problem. Journal of Complexity **86**, 101898 (2025)

[20] Koussa, E., Macario-Rat, G., Patarin, J.: On the complexity of the permuted kernel problem. Cryptology ePrint Archive (2019)

[21] Budroni, A., Defranceschi, M., Pintore, F.: Recursion enabled: Improved cryptanalysis of the permuted kernel problem. In: Bardet, M., Niederhagen, R. (eds.) Post-Quantum Cryptography, pp. 207–241. Springer, Cham (2026)

[22] 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)

[23] 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)

[24] 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

[25] Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020, pp. 677–706. Springer, Cham (2020)

[26] Kales, D., Zaverucha, G.: An attack on some signature schemes constructed from five-pass identification schemes. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) Cryptology and Network Security, pp. 3–22. Springer, Cham (2020)

[27] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J.

ACM **33**(4), 792–807 (1986) https://doi.org/10.1145/6490.6503

[28] Roy, L.: SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022, pp. 657–687. Springer, Cham (2022)

[29] Baum, C., Braun, L., Beullens, W., Delpech de Saint Guilhem, C., Klooß, M., Majenz, C., Mukherjee, S., Orsini, E., Ramacher, S., Rechberger, C., Roy, L., Scholl, P.: FAEST. Technical report, National Institute of Standards and Technology (2024). available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures

[30] 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)

[31] Chou, T.: Representations of Elementary Vectors in VOLE-in-the-head-based Schemes. Cryptology ePrint Archive, Paper 2025/1336 (2025). https://eprint.iacr.org/2025/1336

[32] Chiang, J.H.-y., Damgård, I., Duro, W.R., Engan, S., Kolby, S., Scholl, P.: Post-quantum threshold ring signature applications from vole-in-the-head. In: Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security. CCS '25, pp. 4664–4678. Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3719027.3744854 . https://doi.org/10.1145/3719027.3744854

[33] Feneuil, T., Rivain, M., Vergnaud, D., Warmé-Janville, A.: Threshold Signatures in the Head. Cryptology ePrint Archive, Paper 2026/1125 (2026). https://eprint.iacr.org/2026/1125

[34] 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

[35] Carter, J.L., Wegman, M.N.: Universal classes of hash functions. Journal of Computer and System Sciences **18**(2), 143–154 (1979) https://doi.org/10.1016/0022-0000(79)90044-8

[36] Bierbrauer, J., Johansson, T., Kabatianskii, G., Smeets, B.: On families of hash functions via geometric codes and concatenation. In: Stinson, D.R. (ed.) Advances in Cryptology — CRYPTO' 93, pp. 331–342. Springer, Berlin, Heidelberg (1994)