Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs

Thibauld Feneuil[1,2], Antoine Joux[3], and Matthieu Rivain[1]

[1] CryptoExperts, Paris, France
[2] Sorbonne Université, CNRS, INRIA, Institut de Mathématiques de Jussieu-Paris Rive Gauche, Ouragan, Paris, France
[3] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

{thibauld.feneuil,matthieu.rivain}@cryptoexperts.com
joux@cispa.de

Abstract. Zero-knowledge proofs of knowledge are useful tools to design signature schemes. The ongoing effort to build a quantum computer urges the cryptography community to develop new secure cryptographic protocols based on quantum-hard cryptographic problems. One of the few directions is code-based cryptography for which the strongest problem is the *syndrome decoding* (SD) for random linear codes. This problem is known to be NP-hard and the cryptanalysis state of the art has been stable for many years. A zero-knowledge protocol for this problem was pioneered by Stern in 1993. Since its publication, many articles proposed optimizations, implementation, or variants.

In this paper, we introduce a new zero-knowledge proof for the syndrome decoding problem on random linear codes. Instead of using permutations like most of the existing protocols, we rely on the MPC-in-the-head paradigm in which we reduce the task of proving the low Hamming weight of the SD solution to proving some relations between specific polynomials. Specifically, we propose a 5-round zero-knowledge protocol that proves the knowledge of a vector x such that $y = Hx$ and $\mathrm{wt}(x) \leq w$ and which achieves a soundness error closed to $1/N$ for an *arbitrary* N.

While turning this protocol into a signature scheme, we achieve a signature size of 11-12 KB for 128-bit security when relying on the hardness of the SD problem on binary fields. Using larger fields (like $\mathbb{F}_{2^8}$), we can produce fast signatures of around 8 KB. This allows us to outperform Picnic3 and to be competitive with SPHINCS$^+$, both post-quantum signature candidates in the ongoing NIST standardization effort. Moreover, our scheme outperforms all the existing code-based signature schemes for the common "signature size + public key size" metric.

1 Introduction

Zero-knowledge proofs are an important tool for many cryptographic protocols and applications. Such proofs enable a *prover* to prove a statement by interacting with a *verifier* without revealing anything more than the statement itself. Zero-knowledge proofs find application in many contexts. Thanks to the Fiat-Shamir transform [FS87], we can convert such proofs into signature schemes. In this article, we aim to build an efficient code-based signature scheme using this methodology. To do so, we will focus on the *generic decoding* problem, a.k.a. the (computational) *syndrome decoding* (SD) problem: given a matrix $H \in \mathbb{F}_q^{(m-k)\times m}$ and a vector $y \in \mathbb{F}_q^{m-k}$, recover a *small-weight* vector $x \in \mathbb{F}_q^m$ such that $Hx = y$. For random linear codes –*i.e.* for a random matrix H– this problem is known to be NP-hard and widely believed to be robust for practical sets of parameters.

In a pioneering work from three decades ago, Stern proposed a zero-knowledge protocol to prove the knowledge of a syndrome decoding solution [Ste94]. This protocol achieves a *soundness error* of 2/3 which means that a malicious prover can fool the verifier with a 2/3 probability. Although an arbitrary security of $(2/3)^\tau$ can be achieved by repeating the protocol τ times, the induced communication cost for standard security levels (*e.g.* 128 bits) becomes significant, which is partly due to this high soundness error. Since the work of Stern, a few papers have proposed optimizations and implementations of this protocol (see for instance [Vér96,GG07,AGS11,ACBH13]) but the communication cost was still heavy for random linear codes with standard security levels.

In 2007, Ishai, Kushilevitz, Ostrovsky and Sahai proposed a new technique to build zero-knowledge proofs from secure multi-party computation (MPC) protocols, which is known as the *MPC-in-the-Head* (MPCitH) paradigm [IKOS07]. While this construction was mainly considered of theoretical interest at first, it has been increasingly applied to build practical schemes over the last years. In particular, the Picnic post-quantum signature scheme [CDG$^+$], which is a third-round alternate candidate of the ongoing NIST standardization effort, is based on the MPCitH principle. Recently, new zero-knowledge protocols for the SD problem have been inspired by this principle [GPS21,FJR21,BGKM22]. In particular, these protocols achieve an arbitrary soundness error $1/N$ instead of the $2/3$ (or $1/2$) of Stern protocol and variants. These protocols result in smaller proof/signature sizes at the cost of computational overheads.

Our contribution. In this article, we build a new zero-knowledge protocol to prove the knowledge of a syndrome decoding solution using the MPCitH paradigm. We further turn this protocol into an efficient code-based signature scheme.

While proving that $y = Hx$ is communication-free in this paradigm, the hard part consists in proving that x is a small-weight vector. We propose here an efficient way to prove that $\mathrm{wt}(x) \leq w$ through a multi-party computation which is simulated by the prover ("in her head"). The key idea is to prove the equality $x \circ v = 0$ where $\circ$ is the component-wise multiplication and where the coefficients of the vector v are the evaluations of a polynomial Q of degree w. By definition, v has at most w zero coordinates, so the relation $x \circ v = 0$ proves that x has at most w non-zero coordinates (*i.e.* $\mathrm{wt}(x) \leq w$). In order to prove the latter relation, we use techniques borrowed from the Banquet signature scheme [BdK$^+$21] with further adaptations. To check that all $x_j \cdot v_j$ are equal to zero, we arrange the input x into a polynomial S, provide a product polynomial $F \cdot P$ as part of the witness, and check that $(F \cdot P)(\cdot)$ indeed equals the product of $S(\cdot)$ and $Q(\cdot)$. This can be done efficiently by only verifying a few products of these polynomials evaluated at some random points. However, instead of revealing the multiplication operands like in [BdK$^+$21], we rely on the product checking protocol proposed in [LN17,BN20] and its batch version recently introduced in [KZ21].

Thanks to the Fiat-Shamir transform [FS87], we convert our protocol into a signature scheme. Our scheme outperforms all the existing code-based signatures for the "signature size + public key size" metric. When relying on the hardness of the syndrome decoding problem over $\mathbb{F}_{256}$, our scheme is below 10 KB for this metric, which makes it competitive with Picnic3 [KZ20b] and SPHINCS$^+$ [BHK$^+$19]. Compared to other code-based signature schemes (such as Wave [DST19] and Durandal [ABG$^+$19]), our scheme has the significant advantage of relying on a non-structured NP-hard decoding problem which has been widely studied over the last decades.

To provide more flexibility, we introduce a parameter d in the definition of the syndrome decoding problem. The idea is, instead of having a constraint for the global weight of the secret vector x, to split x into d chunks $x := (x_1 \mid \ldots \mid x_d)$ and to have a constraint on the weight of each chunk. By taking $d = 1$, this *d-split* version is equivalent to the standard syndrome decoding problem. We provide a security reduction from this variant to the standard problem which allows us to compensate the security loss by a slight increase of the parameters. This so-called *d-split syndrome decoding problem* offers us more flexibility to find better size-performance trade-offs for our signature scheme.

Paper organization. The paper is organized as follows: In Section 2, we introduce the necessary background on the syndrome decoding problem, zero-knowledge proofs, and the MPC-in-the-Head paradigm. We present our protocol in Section 3 and the signature scheme obtained through the Fiat-Shamir transform in Section 4. To conclude, we provide implementation results and compare our construction with other signature schemes from the state of the art in Section 5.

2 Preliminaries

Throughout the paper, $\mathbb{F}$ shall denote a finite field. For any vector $x \in \mathbb{F}^m$, the *Hamming weight* of x, denoted $\mathrm{wt}(x)$, is the number of non-zero coordinates of x. For two vectors $x_1 \in \mathbb{F}^{m_1}$ and $x_2 \in \mathbb{F}^{m_2}$, we denote $(x_1 \,|\, x_2) \in \mathbb{F}^{m_1+m_2}$ their concatenation. We denote $\circ$ the component-wise multiplication between two

vectors. For any $m \in \mathbb{N}^*$, the integer set $\{1, \ldots, m\}$ is denoted $[m]$. For a probability distribution D, the notation $s \leftarrow D$ means that s is sampled from D. For a finite set S, the notation $s \leftarrow S$ means that s is uniformly sampled at random from S. When the set S is clear from the context, we sometimes denote $s \leftarrow \$$ for a uniform random sampling of s from S. For an algorithm $\mathcal{A}$, $out \leftarrow \mathcal{A}(in)$ further means that out is obtained by a call to $\mathcal{A}$ on input in (using uniform random coins whenever $\mathcal{A}$ is probabilistic). Along the paper, probabilistic polynomial time is abbreviated PPT.

A function $\mu : \mathbb{N} \to \mathbb{R}$ is said *negligible* if, for every positive polynomial $p(\cdot)$, there exists an integer $N_p > 0$ such that for every $\lambda > N_p$, we have $|\mu(\lambda)| < 1/p(\lambda)$. When not made explicit, a negligible function in λ is denoted $\mathsf{negl}(\lambda)$ while a polynomial function in λ is denoted $\mathsf{poly}(\lambda)$. We further use the notation $\mathsf{poly}(\lambda_1, \lambda_2, ...)$ for a polynomial function in several variables.

Two distributions $\{D_\lambda\}_\lambda$ and $\{E_\lambda\}_\lambda$ indexed by a security parameter λ are (t, ε)-*indistinguishable* (where t and ε are $\mathbb{N} \to \mathbb{R}$ functions) if, for any algorithm $\mathcal{A}$ running in time at most $t(\lambda)$, we have

$$\left| \Pr[\mathcal{A}(x) = 1 \mid x \leftarrow D_\lambda] - \Pr[\mathcal{A}(x) = 1 \mid x \leftarrow E_\lambda] \right| \leq \varepsilon(\lambda) \ .$$

The two distributions are said

- *computationally indistinguishable* if $\varepsilon \in \mathsf{negl}(\lambda)$ for every $t \in \mathsf{poly}(\lambda)$;
- *statistically indistinguishable* if $\varepsilon \in \mathsf{negl}(\lambda)$ for every (unbounded) t;
- *perfectly indistinguishable* if $\varepsilon = 0$ for every (unbounded) t.

2.1 Standard Cryptographic Notions

Definition 1 (Pseudorandom Generator (PRG)). *Let $G : \{0,1\}^* \to \{0,1\}^*$ and let $\ell(\cdot)$ be a polynomial such that for any input $s \in \{0,1\}^\lambda$ we have $G(s) \in \{0,1\}^{\ell(\lambda)}$. Then, G is a (t, ε)-secure pseudorandom generator if the following two conditions hold:*

- *Expansion: $\ell(\lambda) > \lambda$;*
- *Pseudorandomness: the distributions*

$$\{G(s) \mid s \leftarrow \{0,1\}^\lambda\} \quad \textit{and} \quad \{r \mid r \leftarrow \{0,1\}^{\ell(\lambda)}\}$$

are (t, ε)-indistinguishable.

In this paper we shall make use of a *tree PRG* which is a pseudorandom generator that expands a root seed mseed into N subseeds in a structured way. The principle is to label the root of a binary tree of depth $\lceil \log_2 N \rceil$ with mseed. Then, one inductively labels the children of each node with the output of a standard PRG applied to the node's label. The subseeds $(\mathsf{seed}_i)_{i \in [N]}$ are defined as the labels of the N leaves of the tree. A tree PRG makes it possible to reveal all the subseeds but a small subset $E \subset [N]$ by only revealing $|E| \cdot \log(N)$ labels of the tree (which is presumable much smaller than $N - |E|$). The principle is to reveal the labels on the siblings of the paths from the root of the tree to leaves $i \notin E$ (excluding the labels of those paths themselves). Those labels allow the verifier to reconstruct $(\mathsf{seed}_i)_{i \in E}$ while still hiding $(\mathsf{seed}_i)_{i \notin E}$.

Definition 2 (Collision-Resistant Hash Functions). *A family of functions $\{\mathrm{Hash}_k : \{0,1\}^* \to \{0,1\}^{\ell(\lambda)} ; k \in \{0,1\}^{\kappa(\lambda)}\}_\lambda$ indexed by a security parameter λ is collision-resistant if there exists a negligible function ν such that, for any PPT algorithm $\mathcal{A}$, we have*

$$\Pr\left[\begin{array}{c} x \neq x' \\ \cap \ \mathrm{Hash}_k(x) = \mathrm{Hash}_k(x') \end{array} \middle| \begin{array}{l} k \leftarrow \{0,1\}^{\kappa(\lambda)}; \\ (x, x') \leftarrow \mathcal{A}(k) \end{array} \right] \leq \nu(\lambda) \ .$$

We now formally introduce the notion of commitment scheme which is instrumental in many zero-knowledge protocols.

Definition 3 (Commitment Scheme). *A commitment scheme is a triplet of algorithms* $(\mathsf{KeyGen}, \mathrm{Com}, \mathsf{Verif})$ *such that*

– KeyGen *is a PPT algorithm that, on input* 1^λ*, outputs some public parameters* $\mathrm{PP} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ *containing a definition of the message space, the randomness space and the commitment space.*
– Com *is a deterministic polynomial-time algorithm that, on input the public parameters* PP*, a message* x *and the randomness* ρ*, outputs a commitment* c*.*
– Verif *is a deterministic polynomial-time algorithm that, on input the public parameters* PP*, a message* x*, a commitment* c *and the randomness* ρ*, outputs a bit* $b \in \{0,1\}$*.*

In this article, the public parameter input PP will be made implicit in the calls to Com and Verif.

Definition 4 (Correctness Property). *A commitment scheme achieves correctness, if for any message* x *and any randomness* ρ*:*

$$\Pr[\mathsf{Verif}(x, c, \rho) = 1 \mid c \leftarrow \mathrm{Com}(x; \rho)] = 1 \ .$$

Definition 5 (Hiding Property). *A commitment scheme is said computationally (resp. statistically, resp. perfectly) hiding if, for any two messages* x_0 *and* x_1*, the following distributions*

$$\{c \mid c \leftarrow \mathrm{Com}(x_0; \rho), \rho \leftarrow \$\} \text{ and } \{c \mid c \leftarrow \mathrm{Com}(x_1; \rho), \rho \leftarrow \$\}$$

are computationally (resp. statistically, resp. perfectly) indistinguishable.

Definition 6 (Binding Property). *A commitment scheme is binding if there exists a negligible function* ν *such that, for every (PPT) algorithm* $\mathcal{A}$*, we have*

$$\Pr\left[\begin{array}{c} x \neq x' \\ \cap \mathsf{Verif}(\mathrm{PP}, x, c, \rho) = 1 \\ \cap \mathsf{Verif}(\mathrm{PP}, x', c, \rho') = 1 \end{array} \middle| \begin{array}{l} \mathrm{PP} \leftarrow \mathsf{KeyGen}(); \\ (x, x', \rho, \rho', c) \leftarrow \mathcal{A}(\mathrm{PP}) \end{array}\right] \leq \nu(\lambda) \ ,$$

where the probability is taken over the randomness of $\mathcal{A}$ *and* KeyGen*. If we restrict* $\mathcal{A}$ *to being PPT, then the scheme is computationally binding. If the computation time of* $\mathcal{A}$ *is unbounded, then the scheme is statistically binding.*

2.2 Syndrome Decoding Problems

Definition 7 (Syndrome Decoding Problem). *Let* $\mathbb{F}$ *be a finite field. Let* m*,* k *and* w *be positive integers such that* $m > k$ *and* $m > w$*. The syndrome decoding problem with parameters* $(\mathbb{F}, m, k, w)$ *is the following problem:*

Let H*,* x *and* y *be such that:*

1. H *is uniformly sampled from* $\mathbb{F}^{(m-k)\times m}$*,*
2. x *is uniformly sampled from* $\{x \in \mathbb{F}^m : \mathrm{wt}(x) = w\}$*,*
3. y *is defined as* $y := Hx$*.*

From (H, y)*, find* x*.*

In the following, a pair (H, y) generated as in the above definition is called an *instance* of the syndrome decoding problem for parameters $(\mathbb{F}, m, k, w)$. The syndrome decoding problem is known to be NP-hard. For a weight parameter w lower than the Gilbert-Varshamov radius $\tau_{\mathrm{GV}}(m, k)$, which is defined as:

$$w < \tau_{\mathrm{GV}}(m, k) \quad \Leftrightarrow \quad \sum_{j=0}^{w-1} \binom{m}{j} (q-1)^j < q^{m-k} \quad \text{with} \quad q = |\mathbb{F}| \ ,$$

we know that there exists a unique solution x such that $y = Hx$ with overwhelming probability. Otherwise, an instance has several solutions on average.

There exists two main families of algorithms to solve the syndrome decoding problem: the *information set decoding* (ISD) algorithms and *generalized birthday algorithms* (GBA) [TS16,BBC+19]. To obtain a λ-bit

security, the parameters of the syndrome decoding problem are hence chosen in a way to ensure that both kind of algorithms run in time greater than 2^λ.

Instead of working on the standard syndrome decoding problem, we will consider an alternative version that we shall call the *d-split syndrome decoding* problem, where the secret x is split into d chunks of same Hamming weights.

Definition 8 (d-Split Syndrome Decoding Problem). *Let $\mathbb{F}$ be a finite field. Let m, k, w be positive integers such that $m > k$, $m > w$, $d \mid w$ and $d \mid m$. The d-split syndrome decoding problem with parameters $(\mathbb{F}, m, k, w)$ is the following problem:*

Let H, x and y be such that:

1. *H is uniformly sampled from $\mathbb{F}^{(m-k)\times m}$,*
2. *x is uniformly sampled from*

$$\left\{(x_1 \mid \ldots \mid x_d) \in \mathbb{F}^m \ : \ \forall i \in [d],\, x_i \in \mathbb{F}^{m/d},\, \mathrm{wt}(x_i) = \frac{w}{d}\right\} \ ,$$

3. *y is defined as $y := Hx$.*

From (H, y), find x.

By taking $d = 1$, we get the standard syndrome decoding problem. The following theorem gives a way to estimate the difficulty to solve the d-split syndrome decoding problem.

Theorem 1. *Let $\mathbb{F}$ be a finite field. Let m, k, w be positive integers such that $m > k$, $m > w$, $d \mid w$ and $d \mid m$. Let $\mathcal{A}_d$ be an algorithm which solves a random $(\mathbb{F}, m, k, w)$-instance of the d-split syndrome decoding problem in time t with success probability ε_d. Then there exists an algorithm $\mathcal{A}_1$ which solves a random $(\mathbb{F}, m, k, w)$-instance of the standard syndrome decoding problem in time t with probability ε_1, where*

$$\varepsilon_1 \geq \frac{\binom{m/d}{w/d}^d}{\binom{m}{w}} \cdot \varepsilon_d \ .$$

Informally, an instance of the standard syndrome decoding problem is an instance of the d-split syndrom decoding problem with probability $\binom{m/d}{w/d}^d / \binom{m}{w}$. Moreover, a standard syndrome decoding instance can be "randomized" and input to the d-split adversary as much as desired. A formal proof of the above theorem is provided in Appendix A.

2.3 Interactive Protocols

A two-party protocol is a triplet $\Pi = (\mathrm{Init}, \mathcal{A}, \mathcal{B})$ where Init is an initialization algorithm that, on input 1^λ, produces a pair $(in_\mathcal{A}, in_\mathcal{B})$, and where $\mathcal{A}$ and $\mathcal{B}$ are two stateful algorithms, called the *parties*. The parties originally receive their inputs $in_\mathcal{A}$ and $in_\mathcal{B}$ then interacts by exchanging messages, and finally one of the parties, say $\mathcal{B}$, produces the output of the protocol. More formally, an execution of the protocol consists in a sequence:

$\mathsf{state}_\mathcal{A} \leftarrow \mathcal{A}(in_\mathcal{A})$
$\mathsf{state}_\mathcal{B} \leftarrow \mathcal{B}(in_\mathcal{B})$
$(\mathrm{Msg}_\mathcal{A}[0], \mathsf{state}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{state}_\mathcal{A})$
$\vdots$
$(\mathrm{Msg}_\mathcal{B}[i], \mathsf{state}_\mathcal{B}) \leftarrow \mathcal{B}(\mathsf{state}_\mathcal{B}, \mathrm{Msg}_\mathcal{A}[i-1])$
$(\mathrm{Msg}_\mathcal{A}[i], \mathsf{state}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{state}_\mathcal{A}, \mathrm{Msg}_\mathcal{B}[i])$
$\vdots$
$out \leftarrow \mathcal{B}(\mathsf{state}_\mathcal{B}, \mathrm{Msg}_\mathcal{A}[n])$

The sequence of exchanged messages is called the *transcript* of the execution, which is denoted

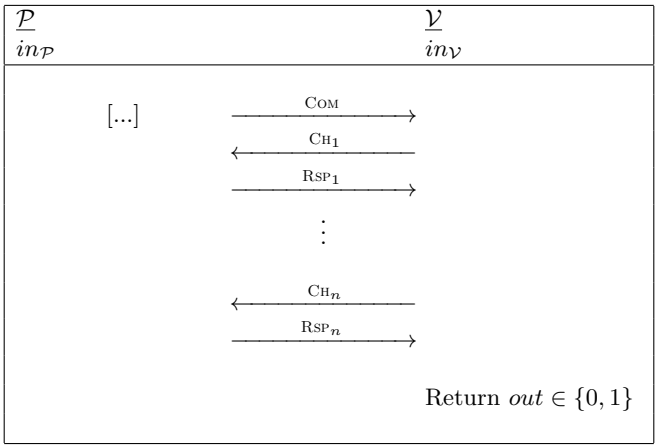
$$\text{View}(\langle \mathcal{A}(in_{\mathcal{A}}), \mathcal{B}(in_{\mathcal{B}}) \rangle) := (\text{Msg}_{\mathcal{A}}[0], \text{Msg}_{\mathcal{B}}[1], \ldots, \text{Msg}_{\mathcal{A}}[n]) \ .$$

An execution producing an output *out* is further denoted

$$\langle \mathcal{A}(in_{\mathcal{A}}), \mathcal{B}(in_{\mathcal{B}}) \rangle \to out \ .$$

In our exposition, the state of the parties shall be made implicit. We shall then say that an algorithm has *rewindable black-box access* to a party $\mathcal{A}$ if this algorithm can copy the state of $\mathcal{A}$ at any moment, relaunch $\mathcal{A}$ from a previously copied state, and query $\mathcal{A}$ (with its current state) on input messages. A variable x is said to be *extractable* from $\mathcal{A}$ if there exists a PPT algorithm $\mathcal{E}$ which, given a rewindable black-box access to $\mathcal{A}$, returns x after a polynomial number of queries to $\mathcal{A}$.

Interactive proofs. We will focus on a special kind of two-party protocol called an *interactive proof* which involves a *prover* $\mathcal{P}$ and a *verifier* $\mathcal{V}$. In such a protocol, $\mathcal{P}$ tries to prove a statement to $\mathcal{V}$. The first message sent by $\mathcal{P}$ is called a *commitment*, denoted Com. From this commitment $\mathcal{V}$ produces a first *challenge* Ch_1 to which $\mathcal{P}$ answers with a response Rsp_1, followed by a next challenge Ch_2 from $\mathcal{V}$, and so on. After receiving the last response Rsp_n, $\mathcal{V}$ produces a binary output: either 1, meaning that she was convinced by $\mathcal{P}$, or 0 otherwise. Such an m-round interactive proof with $m = 2n+1$ (1 commitment + n challenge-response pairs) is illustrated on Protocol 1.



Protocol 1: Structure of a m-round interactive proof with $m = 2n + 1$.

2.4 Zero-Knowledge Proofs of Knowledge

Informally, a proof of knowledge is an interactive proof in which $\mathcal{P}$ aims to convince $\mathcal{V}$ that she knows something. Formally, a proof of knowledge is defined as follows:

Definition 9 (Proof of Knowledge). *Let x be a statement of language L in NP, and $W(x)$ the set of witnesses for x such that the following relation holds:*

$$\mathcal{R} = \{(x, w) : x \in L, w \in W(x)\} \ .$$

A proof of knowledge for relation $\mathcal{R}$ with soundness error ε is a two-party protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with the following two properties:

– *(Perfect) Completeness: If $(x, w) \in \mathcal{R}$, then a prover $\mathcal{P}$ who knows a witness w for x succeeds in convincing the verifier $\mathcal{V}$ of his knowledge. More formally:*

$$\Pr[\langle \mathcal{P}(x, w), \mathcal{V}(x) \rangle \to 1] = 1,$$

i.e. given the interaction between the prover $\mathcal{P}$ and the verifier $\mathcal{V}$, the probability that the verifier is convinced is 1*.*

– *Soundness: If there exists a PPT prover $\tilde{\mathcal{P}}$ such that*

$$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x) \rangle \to 1] > \varepsilon,$$

then there exists an algorithm $\mathcal{E}$ (called an extractor) which, given rewindable black-box access to $\tilde{\mathcal{P}}$, outputs a witness w' for x in time $\mathsf{poly}(\lambda, (\tilde{\varepsilon} - \varepsilon)^{-1})$ with probability at least $1/2$*.*

Informally, a proof of knowledge has soundness error ε if a prover $\tilde{\mathcal{P}}$ without knowledge of the witness cannot convince the verifier with probability greater than ε assuming that the underlying problem (recovering a witness for the input statement) is hard. Indeed, if a prover $\tilde{\mathcal{P}}$ can succeed with a probability greater than ε, then the existence of the extractor (algorithm $\mathcal{E}$) implies that $\tilde{\mathcal{P}}$ can be used to compute a witness $w' \in W(x)$.

Remark 1. In the present article, we focus on proof of knowledge for a syndrome decoding instance defined by a matrix H and a vector y. The problem parameters m, k and w will be considered to be defined by the security parameter λ. In this context, the syndrome decoding instance (H, y) is the *statement*. A *witness* for this statement is a small-weight vector x such that $y = Hx$.

We now recall the notion of honest-verifier zero-knowledge proof:

Definition 10 (Honest-Verifier Zero-Knowledge Proof). *A proof of knowledge is {computationally, statistically, perfectly} honest-verifier zero-knowledge (HVZK) if there exists a PPT algorithm $\mathcal{S}$ (called simulator) whose output distribution is {computationally, statistically, perfectly} indistinguishable from the distribution* $\mathrm{View}(\langle \mathcal{P}(x, w), \mathcal{V}(x) \rangle)$ *obtained with an honest $\mathcal{V}$.*

Informally, the previous definition says a genuine execution of the protocol can be simulated without any knowledge of the witness. In other words, the transcript of an execution between the prover and an honest verifier does not reveal any information about the witness.

2.5 Sharings and Multi-Party Computation

In the scope of this article, all the *sharings* are additive. Specifically, an N-sharing of an element $x \in \mathbb{F}^m$ is an N-tuple

$$[\![x]\!] = \left([\![x]\!]_1, \ldots, [\![x]\!]_N\right) \in (\mathbb{F}^m)^N \text{ such that } x = \sum_{i=1}^{N} [\![x]\!]_i \ .$$

Each $[\![x]\!]_i$ is called a *share* of x. For a polynomial $P \in \mathbb{F}[X]$ of degree at most d, we define its sharing $[\![P]\!]$ as a N-tuple of $(\mathbb{F}[X])^N$ such that $P = \sum_{i=1}^{N} [\![P]\!]_i$, where each $[\![P]\!]_i$ is of degree at most d. In particular, a sharing of a degree-d polynomial can be seen as the sharing of the d-tuple of its coefficients.

In the context of multi-party computation (MPC), an N-sharing is usually distributed to N parties, meaning that each party gets one of the N shares. From those shares, the parties can perform distributed computation. Let assume that each party $i \in [N]$ receives the shares $[\![x]\!]_i$, $[\![y]\!]_i$ and $[\![P]\!]_i$ corresponding to shared values $x, y \in \mathbb{F}$ and polynomial $P \in \mathbb{F}[X]$. They can perform the following operations:

– **Additions:** the parties locally compute $[\![x + y]\!]$ by adding their respective shares:

$$\forall i, [\![x + y]\!]_i := [\![x]\!]_i + [\![y]\!]_i \ .$$

This process is denoted $[\![x + y]\!] = [\![x]\!] + [\![y]\!]$.

– **Addition with a constant:** for a given constant α, the parties locally compute $[\![x+\alpha]\!]$ by doing:

$$\begin{cases} [\![x+\alpha]\!]_1 := [\![x]\!]_1 + \alpha \\ [\![x+\alpha]\!]_i := [\![x]\!]_i \text{ for } i \neq 1 \end{cases}$$

This process is denoted $[\![x+\alpha]\!] = [\![x]\!] + \alpha$.

– **Multiplication by a constant:** for a given constant α, the parties locally compute $[\![\alpha \cdot x]\!]$ by multiplying their respective shares:

$$\forall i, [\![\alpha \cdot x]\!]_i := \alpha \cdot [\![x]\!]_i \ .$$

This process is denoted $[\![\alpha \cdot x]\!] = \alpha \cdot [\![x]\!]$.

– **Polynomial evaluation:** for a given r, the parties can locally compute $[\![P(r)]\!]$ by:

$$\forall i, [\![P(r)]\!]_i := [\![P]\!]_i(r) = \sum_{j=0}^{d} [\![P_j]\!]_i \cdot r^j \ ,$$

where $\{[\![P_j]\!]_i\}_j$ denotes the coefficients of $[\![P]\!]_i$. This process is denoted $[\![P(r)]\!] = [\![P]\!](r)$.

2.6 The MPC-in-the-Head Paradigm

The MPC-in-the-Head (MPCitH) paradigm introduced in [IKOS07] offers a way to build zero-knowledge proofs from secure multi-party computation (MPC) protocols. Let us assume we have an MPC protocol in which N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ securely and correctly evaluate a function f on a secret input x with the following properties:

– the secret x is encoded as a sharing $[\![x]\!]$ and each $\mathcal{P}_i$ takes a share $[\![x]\!]_i$ as input;
– the function f outputs Accept or Reject;
– the views of t parties leak no information about the secret x.

We can use this MPC protocol to build a zero-knowledge proof of knowledge of an x for which $f(x)$ evaluates to Accept. The prover proceeds as follows:

– she builds a random sharing $[\![x]\!]$ of x;
– she simulates locally ("in her head") all the parties of the MPC protocol;
– she sends commitments to each party's view, *i.e.* party's input share, secret random tape and sent and received messages, to the verifier;
– she sends the output shares $[\![f(x)]\!]$ of the parties, which should correspond to Accept.

Then the verifier randomly chooses t parties and asks the prover to reveal their views. After receiving them, the verifier checks that they are consistent with an honest execution of the MPC protocol and with the commitments. Since only t parties are opened, revealed views leak no information about the secret x, while the random choice of the opened parties makes the cheating probability upper bounded by $(N-t)/N$, thus ensuring the soundness of the zero-knowledge proof.

In this article, we shall only consider the case $t = N-1$, *i.e.* when the verifier asks to open all the parties except one. We shall further consider that the function f computed by the MPC protocol might be non-deterministic. Specifically, if the protocol takes what we shall call a *good witness* x as input then the protocol returns Accept with probability 1. Otherwise, the protocol shall reject most of the time but might still accept with some *false positive* probability p. To summarize, we consider a setting in which the output of the protocol has a probability distribution of the form described in Table 1.

While moving to the MPCitH setting, the randomness for f is then provided by the verifier and the "pre-randomness" view of each party (input share, random tape, initial message) must be committed before receiving the randomness from the verifier. If the prover is honest (*i.e.* knows a "good witness" x), it will always convince the verifier. On the other hand, a malicious prover might successfully cheat with probability $1/N$ (by corrupting the computation of one party) or make the MPC protocol produce a false positive with probability p. Thus, the resulting zero-knowledge protocol has a soundness error of

$$1 - \left(1 - \frac{1}{N}\right)(1-p) = \frac{1}{N} + p - \frac{1}{N} \cdot p \ .$$

	Output of f	
	ACCEPT	REJECT
x is a good witness	1	0
x is *not* a good witness	p	$1-p$

Table 1: Probability distribution of the output of the MPC protocol

2.7 Multi-Party Product Verification

A triple of sharings $([\![a]\!], [\![b]\!], [\![c]\!])$ of three elements $a, b, c \in \mathbb{F}$ is called a *multiplication triple* (or Beaver triples [Bea92]) if the shared values satisfy $a \cdot b = c$. The ability to check the correctness of a multiplication triple is instrumental in many MPC (in the Head) protocols.

The authors of [LN17,BN20] propose an MPC protocol to verify the correctness of a multiplication triple by "sacrificing" another one. Specifically, given a random triple $([\![a]\!], [\![b]\!], [\![c]\!])$, the protocol simultaneously verifies the correctness of $([\![x]\!], [\![y]\!], [\![z]\!])$ and $([\![a]\!], [\![b]\!], [\![c]\!])$, *i.e.* verifies that $c = a \cdot b$ and $z = x \cdot y$, without revealing any information on (x, y, z) in the following way:

1. The parties get a random $\varepsilon \in \mathbb{F}$ (from the verifier in the MPCitH paradigm),
2. The parties locally set $[\![\alpha]\!] = \varepsilon[\![x]\!] + [\![a]\!]$ and $[\![\beta]\!] = [\![y]\!] + [\![b]\!]$.
3. The parties broadcast $[\![\alpha]\!]$ and $[\![\beta]\!]$ to obtain α and β.
4. The parties locally set $[\![v]\!] = \varepsilon[\![z]\!] - [\![c]\!] + \alpha \cdot [\![b]\!] + \beta \cdot [\![a]\!] - \alpha \cdot \beta$.
5. The parties broadcast $[\![v]\!]$ to obtain v.
6. The parties output ACCEPT if $v = 0$ and REJECT otherwise.

Observe that if both triples are correct multiplication triples (*i.e.*, $z = xy$ and $c = ab$) then the parties will always accept since

$$\begin{aligned} v &= \varepsilon \cdot z - c + \alpha \cdot b + \beta \cdot a - \alpha \cdot \beta \\ &= \varepsilon \cdot x \cdot y - a \cdot b + (\varepsilon \cdot x + a) \cdot b + (y + b) \cdot a - (\varepsilon \cdot x + a) \cdot (y + b) = 0 \end{aligned}$$

In contrast, if one or both triples are incorrect, then the parties will accept with probability at most $1/|\mathbb{F}|$ as shown in Lemma 1.

Lemma 1 ([BN20]). *If $([\![a]\!], [\![b]\!], [\![c]\!])$ or $([\![x]\!], [\![y]\!], [\![z]\!])$ is an incorrect multiplication triple then the parties output* ACCEPT *in the sub-protocol above with probability $\frac{1}{|\mathbb{F}|}$.*

The authors of [KZ21] propose a variant of the above protocol to batch the verification of the d multiplication triples $([\![x_j]\!], [\![y_j]\!], [\![z_j]\!])$ by sacrificing a random *dot-product tuple* $(([\![a_j]\!], [\![b_j]\!])_{j\in[d]}, [\![c]\!])$ verifying $c = \langle a, b \rangle$.

1. The parties gets a random $\varepsilon \in \mathbb{F}^d$ (from the verifier in the MPCitH paradigm),
2. The parties locally set $[\![\alpha]\!] = \varepsilon \circ [\![x]\!] + [\![a]\!]$ and $[\![\beta]\!] = [\![y]\!] + [\![b]\!]$.
3. The parties broadcast $[\![\alpha]\!]$ and $[\![\beta]\!]$ to obtain α and β.
4. The parties locally set $[\![v]\!] = -[\![c]\!] + \langle \varepsilon, [\![z]\!] \rangle + \langle \alpha, [\![b]\!] \rangle + \langle \beta, [\![a]\!] \rangle - \langle \alpha, \beta \rangle$.
5. The parties broadcast $[\![v]\!]$ to obtain v.
6. The parties output ACCEPT if $v = 0$ and REJECT otherwise.

Lemma 2 ([KZ21]). *If $([\![x_j]\!], [\![y_j]\!], [\![z_j]\!])_{j\in[d]}$ contains an incorrect multiplication triple or if $(([\![a_j]\!], [\![b_j]\!])_{j\in[d]}, [\![c]\!])$ form an incorrect dot product, then the parties output* ACCEPT *in the sub-protocol above with probability at most $\frac{1}{|\mathbb{F}|}$.*

This variant requires less communication for c and v, compared to the case where we repeat d times the original protocol. But depending on the context, repeating d times the original protocol might be preferred to lower the false positive probability (*i.e.* $1/|\mathbb{F}|^d$ against $1/|\mathbb{F}|$).

3 A Zero-Knowledge Protocol for Syndrome Decoding

Let us consider an instance (H, y) of the (d-split) syndrome decoding problem, and let us denote x a solution of this instance. We denote $\mathbb{F}_{\mathrm{SD}}$ the field on which the instance is defined.

Without loss of generality, we assume that H is in the standard form, *i.e.* that $H = (H'|I_{m-k})$ for some $H' \in \mathbb{F}_{\mathrm{SD}}^{(m-k)\times k}$. Thus the solution x can be written as $(x_A|x_B)$ such that we have the linear relation

$$y = H'x_A + x_B \ . \tag{1}$$

This implies that one simply needs to send x_A ($k \cdot \log |\mathbb{F}_{\mathrm{SD}}|$ bits) to reveal the solution of the instance (H, y).

In the following sections, we first build an MPC protocol that takes a sharing of $[\![x_A]\!]$, builds the corresponding $[\![x]\!]$ thanks to Equation (1), and checks that $[\![x]\!]$ corresponds to a vector with a Hamming weight of at most w/d on each chunk. Since $[\![x]\!]$ would verify $y = Hx$ by construction, this MPC protocol verifies that $[\![x_A]\!]$ corresponds to a solution of the syndrome decoding instance (H, y). Then, in Section 3.3, we transform it into a zero-knowledge protocol which proves the knowledge of a solution of the syndrome decoding instance (H, y) thanks to the MPC-in-the-Head paradigm (described in Section 2.6).

3.1 Standard Case ($d = 1$)

We first focus on the case where (H, y) is an instance of the standard syndrome decoding problem (*i.e.* we have $d = 1$). We will then show how to extend the protocol to the general case of any d. We consider a field extension $\mathbb{F}_{\mathrm{poly}} \supseteq \mathbb{F}_{\mathrm{SD}}$ such that $|\mathbb{F}_{\mathrm{poly}}| \geq m$ (we recall that m is the length of the secret x, *i.e.* $x \in \mathbb{F}_{\mathrm{SD}}^m$). We denote $\phi : \mathbb{F}_{\mathrm{SD}} \to \mathbb{F}_{\mathrm{poly}}$ the canonical inclusion of $\mathbb{F}_{\mathrm{SD}}$ into $\mathbb{F}_{\mathrm{poly}}$. Let us take a bijection γ between $\{1, \ldots, |\mathbb{F}_{\mathrm{poly}}|\}$ and $\mathbb{F}_{\mathrm{poly}}$. Then, to ease the notation, we denote γ_i for $\gamma(i)$.

The protocol must check that $y = Hx$ and $\mathrm{wt}(x) \leq w$. As explained in the introduction of the section, the input for the MPC protocol will be $[\![x_A]\!]$, then it will build the sharing $[\![x]\!]$ using the linear relation (1). Then we directly have that $y = Hx$. It remains to check that $\mathrm{wt}(x) \leq w$.

To prove that $\mathrm{wt}(x) \leq w$, the prover build the three following polynomials:

- The polynomial $S \in \mathbb{F}_{\mathrm{poly}}[X]$ satisfying

  $$\forall i \in [m],\ S(\gamma_i) = \phi(x_i) \ ,$$

  as well as $\deg S \leq m - 1$. This S is unique and can be computed by interpolation.
- The polynomial $Q \in \mathbb{F}_{\mathrm{poly}}[X]$ defined as

  $$Q(X) := \prod_{i \in E} (X - \gamma_i)$$

  for some $E \subset [m]$ such that $|E| = w$ and $\{i \in [m] \ : \ x_i \neq 0\} \subset E$, implying $\deg Q = w$.
- The polynomial $P \in \mathbb{F}_{\mathrm{poly}}[X]$ defined as

  $$P := (Q \cdot S)/F \ \text{ with } \ F(X) := \prod_{i=1}^{m} (X - \gamma_i) \ .$$

We stress some useful properties of these polynomials:

- The polynomial Q is a monic polynomial of degree w. Moreover, for every $i \in [m]$, we have

  $$x_i \neq 0 \ \Rightarrow \ i \in E \ \Rightarrow \ Q(\gamma_i) = 0 \ .$$
- The polynomial F divides $Q \cdot S$. Indeed, for every $i \in [m]$, we have

  $$(Q \cdot S)(\gamma_i) = 0$$

  since $S(\gamma_i) \neq 0 \ \Rightarrow \ x_i \neq 0 \ \Rightarrow \ Q(\gamma_i) = 0$. The polynomial P is hence well defined.

– The polynomial P has degree $\deg P \leq w-1$.

If the prover convinces the verifier that there exists two polynomials P (with $\deg P \leq w-1$) and Q (with $\deg Q = w$) such that $Q \cdot S - P \cdot F = 0$ where S and F are built as described above, then the verifier can deduce the following:

$$\begin{aligned} &\forall i \in [m],\ (Q \cdot S)(\gamma_i) = P(\gamma_i) \cdot F(\gamma_i) = 0 \\ &\qquad \Rightarrow\ \forall i \in [m],\ Q(\gamma_i) = 0 \ \text{ or } \ S(\gamma_i) = \phi(x_i) = 0 \end{aligned}$$

Since Q has at most w roots, the verifier concludes that $\phi(x_i) \neq 0$ in at most w positions. Thus $\mathrm{wt}(x) \leq w$.

We now explain how to prove this statement in the MPCitH paradigm. For this purpose, we describe an MPC protocol, which on input x, P and Q outputs Accept if the above condition is verified and Reject otherwise, except with a small false positive probability. The parties' inputs are defined as the shares of $[\![x_A]\!]$, $[\![Q]\!]$ and $[\![P]\!]$. Let us recall that a sharing of a polynomial is naturally defined as a sharing of its coefficients (see Section 2.5). However, for the sharing of Q, we share all of its coefficients except the leading one. Indeed since Q is monic, its leading coefficient is publicly known and is equal to 1. Moreover, it enables to convince the verifier that Q is of degree *exactly* w, which is important since otherwise, a malicious prover could take Q as the zero polynomial.

From its inputs, the MPC protocol first builds the polynomial S from x_A. Then, to verify $Q \cdot S = P \cdot F$, it evaluates the two sides of the relation on t random points r_1, ..., r_t (sampled by the verifier in the MPCitH setting). If the relation is not verified, the probability to observe $Q(r_j) \cdot S(r_j) = P(r_j) \cdot F(r_j)$ for all $j \in [t]$ will be low, which stems from the Schartz-Zippel Lemma (see Appendix C). The larger the set from which the evaluation points r_j are sampled, the smaller the false positive probability p. For this reason, we take these evaluation points in a field extension $\mathbb{F}_{\text{points}}$ of $\mathbb{F}_{\text{poly}}$. Such a field extension allows us to have more points and so to detect more efficiently when $Q \cdot S \neq P \cdot F$. In practice, given an evaluation point r_j, the parties of the MPC protocol verify the relations $Q(r_j) \cdot S(r_j) = (P \cdot F)(r_j)$ by sacrificing multiplication triples as described in Section 2.7. To proceed, the prover must previously build t multiplication triples $([\![a_j]\!], [\![b_j]\!], [\![c_j]\!])$ for random elements $a_j, b_j, c_j \in \mathbb{F}_{\text{points}}$ satisfying $a_j \cdot b_j = c_j$ for $j \in [t]$ and include them to the parties' inputs (each party getting its corresponding share from $[\![a_j]\!]$, $[\![b_j]\!]$ and $[\![c_j]\!]$).

The MPC protocol runs as follows:

1. The parties sample t random points $r_1, \ldots, r_t$ of $\mathbb{F}_{\text{points}}$.
2. The parties locally compute $[\![x]\!]$ from $[\![x_A]\!]$ using Equation (1).
3. The parties locally compute $[\![S(r_j)]\!]$, $[\![Q(r_j)]\!]$ and $[\![(F \cdot P)(r_j)]\!]$ for all $j \in [t]$. Let us remark that $[\![S(r_j)]\!]$ can be computed from $[\![x]\!]$ by the parties without any interaction thanks to the linearity of Lagrange interpolation formula:
$$[\![S(r_j)]\!] = \sum_{i \in [m]} [\![x_i]\!] \prod_{\ell \in [m], \ell \neq i} \frac{r_j - \gamma_\ell}{\gamma_i - \gamma_\ell}\ .$$
On the other hand $[\![(F \cdot P)(r_j)]\!]$ is computed as $F(r_j) \cdot [\![P(r_j)]\!]$ since F is publicly known.
4. For every $j \in [t]$, the parties run an MPC verification of the multiplication triple $\big([\![S(r_j)]\!], [\![Q(r_j)]\!], [\![(F \cdot P)(r_j)]\!]\big)$ by sacrificing the triple $([\![a_j]\!], [\![b_j]\!], [\![c_j]\!])$:
   – The parties sample a random $\varepsilon_j \in \mathbb{F}_{\text{points}}$.
   – The parties locally set
   $$[\![\alpha_j]\!] = \varepsilon_j \cdot [\![Q(r_j)]\!] + [\![a_j]\!] \ \text{ and } \ [\![\beta_j]\!] = [\![S(r_j)]\!] + [\![b_j]\!].$$
   – The parties broadcast $[\![\alpha_j]\!]$ and $[\![\beta_j]\!]$ to obtain α_j and β_j.
   – The parties locally set
   $$[\![v_j]\!] = \varepsilon_j \cdot [\![(F \cdot P)(r_j)]\!] - [\![c_j]\!] + \alpha_j \cdot [\![b_j]\!] + \beta_j \cdot [\![a_j]\!] - \alpha_j \cdot \beta_j\ .$$
   – The parties broadcast $[\![v_j]\!]$ to obtain v_j.

5. The parties output ACCEPT if $v = 0$ and REJECT otherwise.

Note that we do not need to specify how the random values r_j's and ε_j's are sampled by the parties since they will be provided as challenges from the verifier while turning to the zero-knowledge setting.

The above MPC protocol computes a non-deterministic function f which takes x, Q and P (and t multiplication triples) as input and which outputs ACCEPT or REJECT. The randomness of this function comes from the random evaluations points $r_1, \ldots, r_t$ and from the random challenges $\varepsilon_1, \ldots, \varepsilon_t$ used by the product checking protocol. Whenever x indeed satisfies $\mathrm{wt}(x) \leq w$ and the polynomials P and Q are genuinely computed as described above, the protocol outputs ACCEPT with probability one. Whenever the protocol input is not of this form, the protocol shall output REJECT except with a small false positive probability p. In other words, the output of the above protocol follows the distribution depicted in Table 1 where a good witness here means an x of weight at most w and polynomials P and Q which are correctly built.

Let us explicit the false positive probability p. We shall denote $\Delta := |\mathbb{F}_{\text{points}}|$. Whenever the protocol input is not a good witness, *i.e.* $\mathrm{wt}(x) > w$, P or Q are not correctly built, we have $Q \cdot S \neq F \cdot P$. In the above protocol, both sides of the relation are evaluated in t random points. The probability to have the equality for i evaluation points among the t points is at most

$$\frac{\max_{\ell \leq m+w-1} \left\{ \binom{\ell}{i} \binom{\Delta-\ell}{t-i} \right\}}{\binom{\Delta}{t}}$$

since $Q \cdot S - F \cdot P$ is a polynomial of degree at most $m + w - 1$. This holds from a simple extension of the Schwartz-Zippel Lemma that we provide in Appendix C. When this event occurs, the probability to obtain ACCEPT as output is

$$\left(\frac{1}{\Delta}\right)^{t-i},$$

which corresponds to the probability to get the $t-i$ false positives in the verification of multiplication triples (for the $t-i$ remaining evaluation points r_j for which $Q(r_j) \cdot S(r_j) \neq F(r_j) \cdot P(r_j)$). Thus, the global false positive probability p satisfies

$$p \leq \sum_{i=0}^{t} \frac{\max_{\ell \leq m+w-1} \left\{ \binom{\ell}{i} \binom{\Delta-\ell}{t-i} \right\}}{\binom{\Delta}{t}} \left(\frac{1}{\Delta}\right)^{t-i}. \tag{2}$$

3.2 General case (any d)

Let us now assume that (H, y) is an instance of a d-split syndrome decoding problem for some $d \geq 1$. We can easily adapt our protocol in that case. Instead of having a unique polynomial Q of degree w, we will have d polynomials $Q_1, \ldots, Q_d$ of degree exactly w/d to prove the weight bound $\mathrm{wt}(x_j) \leq w/d$ for each chunk x_j of the SD solution. We then have d polynomials S_j (of degree $m/d - 1$) and d polynomials P_j (of degree $w/d - 1$) satisfying the d relations $Q_j \cdot S_j = F \cdot P_j$ with $F := \prod_{j=1}^{m/d}(X - \gamma_j)$. To prove those d relations we evaluate each of them on t random points $r_1, \ldots, r_t$. We stress that the same t random points can be used for each chunk, *i.e.* for every $j \in [d]$.

A malicious prover might try to cheat on a single relation (*i.e.* on a single chunk of the SD solution), in such a way that there exists $j_0 \in [d]$ with

$$\begin{cases} Q_{j_0} \cdot S_{j_0} \neq F \cdot P_{j_0}, \\ \forall j \neq j_0,\ Q_j \cdot S_j = F \cdot P_j. \end{cases}$$

So for a given point r, we use the dot-product checking of [KZ21] (described in Section 2.7) to check all the equalities $Q_j(r) \cdot S_j(r) = F(r) \cdot P_j(r)$ at once. This saves communication without impacting the soundness error compared to independent checks of the d relations.

Whenever the input x, $\{P_j\}$, $\{Q_j\}$ is not a good witness (*i.e.* whenever one x_j has a weight greater than w/d or one polynomial P_j or Q_j is not correctly built), at least one of the relations $Q_j \cdot S_j = F \cdot P_j$ is not verified. Since $Q_j \cdot S_j - F \cdot P_j$ is a polynomial of degree at most $(m+w)/d - 1$, the global false positive probability for the d-split variant becomes

$$p \leq \sum_{i=0}^{t} \frac{\max_{\ell \leq (m+w)/d-1} \left\{ \binom{\ell}{i} \binom{\Delta-\ell}{t-i} \right\}}{\binom{\Delta}{t}} \left(\frac{1}{\Delta}\right)^{t-i} \tag{3}$$

with $\Delta := |\mathbb{F}_{\text{points}}|$. (This upper bound is equivalent to (2) where the max degree $m+w-1$ is replaced by $(m+w)/d-1$).

The constraint on the size of $\mathbb{F}_{\text{poly}}$ now becomes

$$|\mathbb{F}_{\text{poly}}| \geq \frac{m}{d}$$

since we only need w/d points for the interpolation of the polynomials $S_1, \dots S_d$. Thus using the d-split version allows us to use smaller fields for $\mathbb{F}_{\text{poly}}$ and $\mathbb{F}_{\text{points}}$.

Let us note that in practice the new communication is not smaller than before, but rather equivalent or higher, since we need to use bigger syndrome decoding instances to compensate the security loss of the d-split version. The main benefit to introduce the d-split version is to work on polynomials of smaller degree and/or on specific fields which provides better performance trade-offs (see Section 4.5).

3.3 Description of the Protocol

We now give the formal description of our zero-knowledge protocol (general case) in Protocol 2. For the sake of clarity in the protocol description, we denote $\boldsymbol{Q}$ the tuple of polynomials $(Q_1, \dots, Q_d)$. Same for the polynomials $\boldsymbol{P}$ and $\boldsymbol{S}$. The additions, substractions and polynomial evaluations of these tuples are component-wise defined. For example, for a point $r \in \mathbb{F}_{\text{points}}$, $\boldsymbol{Q}(r)$ means $(Q_1(r), \dots, Q_d(r))$. We also use this bold notation for $\boldsymbol{a}_j$, $\boldsymbol{b}_j$, $\boldsymbol{\alpha}_j$, $\boldsymbol{\beta}_j$ and $\boldsymbol{\varepsilon}_j$ which shall represent vectors of $\mathbb{F}_{\text{points}}^d$. Let us recall that $\circ$ denotes the component-wise multiplication. In the scope of this protocol, the polynomial F is defined as $F(X) := \prod_{i=1}^{m/d}(X - \gamma_i)$ with $\mathbb{F}_{\text{poly}} = \{\gamma_1, \gamma_2, \dots\}$.

3.4 Security Proofs

The following theorems state the completeness, zero-knowledge and soundness of Protocol 2. The proofs of Theorems 3 and 4 are provided in appendices E and F.

Theorem 2 (Completeness). *Protocol 2 is perfectly complete, i.e. a prover $\mathcal{P}$ who knows a solution x to the syndrome decoding instance (H, y) and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*

Proof. For any sampling of the random coins of $\mathcal{P}$ and $\mathcal{V}$, if the computation described in Protocol 2 is genuinely performed then all the checks of $\mathcal{V}$ pass. □

Theorem 3 (Honest-Verifier Zero-Knowledge). *Let the PRG used in Protocol 2 be (t, ε_{PRG})-secure and the commitment scheme* Com *be (t, ε_{Com})-hiding. There exists an efficient simulator $\mathcal{S}$ which, given random challenge i^* outputs a transcript which is $(t, \varepsilon_{PRG} + \varepsilon_{Com})$-indistinguishable from a real transcript of Protocol 2.*

Theorem 4 (Soundness). *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input (H, y), convinces the honest verifier $\mathcal{V}$ on input H, y to accept with probability*

$$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle (H, y) \to 1] > \varepsilon$$

Inputs: Both parties have $H = (H'|I_{m-k}) \in \mathbb{F}_{\mathrm{SD}}^{(m-k)\times m}$ and $y \in \mathbb{F}_{\mathrm{SD}}^{m-k}$, the prover also holds $x := (x_1 \mid x_2 \mid \ldots \mid x_d) \in \mathbb{F}_{\mathrm{SD}}^{m}$ such that $y = Hx$ and $\mathrm{wt}(x_j) \leq w$ for $j \in [d]$.

Round 1: The prover computes the proof witness: for all chunk $j \in [d]$,

1. Choose a set $E \subset [\frac{m}{d}]$ s.t. $|E| = \frac{w}{d}$ and $\{\ell : (x_j)_\ell \neq 0\} \subset E$.
2. Compute $Q_j(X) = \prod_{\ell \in E}(X - \gamma_\ell) \in \mathbb{F}_{\mathrm{poly}}[X]$.
3. Compute $S_j(X) \in \mathbb{F}_{\mathrm{poly}}[X]$ by interpolation *s.t.* $\deg S_j \leq \frac{m}{d} - 1$ and $\forall \ell \in [\frac{m}{d}], S_j(\gamma_\ell) = (x_j)_\ell$.
4. Compute $P_j(X) = S_j(X)Q_j(X)/F(X) \in \mathbb{F}_{\mathrm{poly}}[X]$.

Then, the prover prepares the inputs for the multi-party computation as follows:

1. Sample a root seed: $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$.
2. Compute parties' seeds and commitment randomness $(\mathsf{seed}_i, \rho_i)_{i\in[N]}$ with TreePRG(seed).
3. For each party $i \in \{1, \ldots, N\}$,
   - $[\![\boldsymbol{a}_j]\!]_i, [\![\boldsymbol{b}_j]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$, for each $j \in [t]$
   - If $i \neq N$,
     - $\{[\![c_j]\!]_i\}_{j\in[t]}, [\![x_A]\!]_i, [\![\boldsymbol{Q}]\!]_i, [\![\boldsymbol{P}]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
     - $\mathsf{state}_i = \mathsf{seed}_i$
   - Else,
     - $[\![x_A]\!]_N = x_A - \sum_{\ell\neq N}[\![x_A]\!]_\ell$
     - $[\![\boldsymbol{Q}]\!]_N = \boldsymbol{Q} - \sum_{\ell\neq N}[\![\boldsymbol{Q}]\!]_\ell$.
     - $[\![\boldsymbol{P}]\!]_N = \boldsymbol{P} - \sum_{\ell\neq N}[\![\boldsymbol{P}]\!]_\ell$.
     - $[\![c_j]\!]_N = \langle \boldsymbol{a}_j, \boldsymbol{b}_j\rangle - \sum_{\ell\neq N}[\![c_j]\!]_\ell$, for each $j \in [t]$
     - $\mathsf{aux} = ([\![x_A]\!]_N, [\![\boldsymbol{Q}]\!]_N, [\![\boldsymbol{P}]\!]_N, \{[\![c_j]\!]_N\}_{j\in[t]})$
     - $\mathsf{state}_N = \mathsf{seed}_N \mid\mid \mathsf{aux}$
   - Commit the party's state: $\mathsf{com}_i = \mathrm{Com}(\mathsf{state}_i; \rho_i)$.

The prover builds $h = \mathrm{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$ and sends it to the verifier.

Round 2: The verifier uniformly samples, for each $j \in [t]$, an evaluation point $r_j \leftarrow \mathbb{F}_{\mathrm{points}}$ and a vector $\boldsymbol{\varepsilon}_j \leftarrow \mathbb{F}_{\mathrm{points}}^d$, and sends them to the prover.

Round 3: The prover simulates the MPC protocol:

1. The parties locally set $[\![x_B]\!] = y - H'[\![x_A]\!]$.
2. The parties locally compute $[\![\boldsymbol{S}]\!]$ by interpolation using $[\![x]\!] := ([\![x_A]\!] \mid [\![x_B]\!])$.
3. Then for all $j \in [t]$,
   - The parties locally compute $[\![\boldsymbol{S}(r_j)]\!]$, $[\![\boldsymbol{Q}(r_j)]\!]$ and $[\![\boldsymbol{P}(r_j)]\!]$.
   - They locally set $[\![\boldsymbol{\alpha}_j]\!] = \boldsymbol{\varepsilon}_j \circ [\![\boldsymbol{Q}(r_j)]\!] + [\![\boldsymbol{a}_j]\!]$.
   - They locally set $[\![\boldsymbol{\beta}_j]\!] = [\![\boldsymbol{S}(r_j)]\!] + [\![\boldsymbol{b}_j]\!]$.
   - The parties open $[\![\boldsymbol{\alpha}_j]\!]$ and $[\![\boldsymbol{\beta}_j]\!]$ to get $\boldsymbol{\alpha}_j$ and $\boldsymbol{\beta}_j$.
   - The parties locally set

$$[\![v_j]\!] = -[\![c_j]\!] + \langle \boldsymbol{\varepsilon}_j, F(r_j) \cdot [\![\boldsymbol{P}(r_j)]\!]\rangle + \langle \boldsymbol{\alpha}_j, [\![\boldsymbol{b}_j]\!]\rangle + \langle \boldsymbol{\beta}_j, [\![\boldsymbol{a}_j]\!]\rangle - \langle \boldsymbol{\alpha}_j, \boldsymbol{\beta}_j\rangle \ .$$

The prover builds $h' = \mathrm{Hash}([\![\boldsymbol{\alpha}_1]\!], [\![\boldsymbol{\beta}_1]\!], [\![v_1]\!], \ldots, [\![\boldsymbol{\alpha}_t]\!], [\![\boldsymbol{\beta}_t]\!], [\![v_t]\!])$ and sends it to the verifier.

Round 4: The verifier uniformly samples $i^* \leftarrow [N]$ and sends it to the prover.

Round 5: The prover sends $(\mathsf{state}_i, \rho_i)_{i\neq i^*}$, com_{i^*}, $\{[\![\boldsymbol{\alpha}_j]\!]_{i^*}\}_{j\in[t]}$ and $\{[\![\boldsymbol{\beta}_j]\!]_{i^*}\}_{j\in[t]}$.

Verification: The verifier accepts iff all the following checks succeed:

1. For each $i \neq i^*$, she computes all the commitments to the parties' states: $\mathsf{com}_i = \mathrm{Com}(\mathsf{state}_i; \rho_i)$. Then she checks that $h \stackrel{?}{=} \mathrm{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$.
2. Using $\{\mathsf{state}_i\}_{i\neq i^*}$, she simulates all the parties except for i^*. From the recomputed shares, she checks that $h' \stackrel{?}{=} \mathrm{Hash}([\![\boldsymbol{\alpha}_1]\!], [\![\boldsymbol{\beta}_1]\!], [\![v_1]\!], \ldots, [\![\boldsymbol{\alpha}_t]\!], [\![\boldsymbol{\beta}_t]\!], [\![v_t]\!])$ where $[\![v_j]\!]_{i^*} := -\sum_{i\neq i^*}[\![v_j]\!]_i$.

Protocol 2: Zero-knowledge proof for syndrome decoding.

where the soundness error ε is equal to

$$p + \frac{1}{N} - p \cdot \frac{1}{N}$$

with p defined in Equation (3). Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces with either a witness x such that $y = Hx$ and $\mathrm{wt}(x) \leq w$, or a

commitment collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by

$$\frac{4}{\tilde{\varepsilon}-\varepsilon}\cdot\left(1+\tilde{\varepsilon}\cdot\frac{2\cdot\ln(2)}{\tilde{\varepsilon}-\varepsilon}\right).$$

By adapting the parameters t and Δ, we can produce a protocol with soundness error arbitrarily close to $1/N$.

3.5 Performances

In the following analysis, we exclude the challenges from the communication cost since they are of very moderate impact (and do not count whenever making the protocol non-interactive). The communication then consists into

- $\textsc{Com} := h$,
- $\textsc{Res}_1 := h'$ and
- $\textsc{Res}_2 := \big((\mathsf{state}_i,\rho_i)_{i\neq i^*},\ \mathsf{com}_{i^*},\ \{[\![\boldsymbol{\alpha}_j]\!]_{i^*}\}_{j\in[t]},\ \{[\![\boldsymbol{\beta}_j]\!]_{i^*}\}_{j\in[t]}\big)$.

For $i\neq N$, state_i simply consists in a seed of λ bits. For $i=N$, state_i contains

- a seed of λ bits,
- the share $[\![x_A]\!]_N$ of a plaintext,
- the shares $[\![\boldsymbol{Q}]\!]_N$ and $[\![\boldsymbol{P}]\!]_N$ which are $2\cdot d$ polynomials of degree $w/d-1$,
- and the shares $\{[\![c_j]\!]_N\}_{j\in[t]}$ of t points of $\mathbb{F}_{\text{points}}$.

Let us recall that seeds are sampled using a tree PRG (as defined in Section 2.1). Instead to sending the $N-1$ seeds and commitment randomness of $(\mathsf{state}_i,\rho_i)_{i\neq i^*}$, we can instead send the sibling path from $(\mathsf{state}_{i^*},\rho_{i^*})$ to the tree root, it costs at most $\lambda\cdot\log_2(N)$ bits (we need to reveal $\log_2(N)$ nodes of the tree). Moreover com_{i^*} is a commitment of 2λ bits, and $\{[\![\boldsymbol{\alpha}_j]\!]_{i^*}\}_{j\in[t]},\{[\![\boldsymbol{\beta}_j]\!]_{i^*}\}_{j\in[t]}$ are elements of $\mathbb{F}_{\text{points}}$. The communication cost (in bits) of the protocol is then

$$\textsc{Size} = 4\lambda+\underbrace{k\cdot\log_2|\mathbb{F}_{\text{SD}}|}_{[\![x_A]\!]_N}+\underbrace{(2\cdot w)\cdot\log_2|\mathbb{F}_{\text{poly}}|}_{[\![\boldsymbol{Q}]\!]_N,[\![\boldsymbol{P}]\!]_N}+\underbrace{(2\cdot d+1)\cdot t\cdot\log_2|\mathbb{F}_{\text{points}}|}_{\{[\![\boldsymbol{\alpha}_j]\!]_{i^*},[\![\boldsymbol{\beta}_j]\!]_{i^*},[\![c_j]\!]_N\}_{j\in[t]}}+\underbrace{\lambda\cdot\log_2(N)}_{(\mathsf{seed}_i)_{i\neq i^*}}+\underbrace{2\lambda}_{\mathsf{com}_{i^*}}$$

As usual, to achieve a targeted soundness error $2^{-\lambda}$, we can perform τ parallel repetitions of the protocol such that $\varepsilon^\tau\leq 2^{-\lambda}$. And instead of sending τ values for h and h', we can merge them together to send a single h and a single h'. The communication cost (in bits) of the protocol with τ repetitions is

$$\textsc{Size} = 4\lambda+\tau\cdot\Big(k\cdot\log_2|\mathbb{F}_{\text{SD}}|+(2\cdot w)\cdot\log_2|\mathbb{F}_{\text{poly}}|+(2\cdot d+1)\cdot t\cdot\log_2|\mathbb{F}_{\text{points}}|+\lambda\cdot\log_2(N)+2\lambda\Big)$$

and the obtained soundness error is

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

3.6 Comparison

We compare our new protocol with existing zero-knowledge protocols for syndrome decoding (or equivalently for *message decoding*). We compare these protocols on two SD instances of 128-bit security:

- Instance 1 [FJR21]: Syndrome Decoding on $\mathbb{F}_2$ with parameters

$$(m,k,w)=(1280,640,132);$$

– Instance 2 [CVE11]: Syndrome Decoding on $\mathbb{F}_{2^8}$ with parameters

$$(m, k, w) = (208, 104, 78).$$

The comparison for a soundness error of 2^{-128} is given in the Table 2. For our protocol, we provide two instantiations for each syndrome decoding instance to give the reader an idea of the obtained performance while changing the number of parties. The first instantiation called "short" corresponds to an instantiation which provides small communication cost. The second one called "fast" corresponds to an instantiation with faster computation but higher communication cost. The used parameters $(N, \tau, |\mathbb{F}_{\text{poly}}|, |\mathbb{F}_{\text{points}}|, t)$ for our scheme are

– Instance 1:
  Short: $(256, 16, 2^{11}, 2^{22}, 2) \Rightarrow \varepsilon^\tau = 2^{-128.0}$
  Fast: $(32, 26, 2^{11}, 2^{22}, 1) \Rightarrow \varepsilon^\tau = 2^{-129.6}$
– Instance 2:
  Short: $(256, 16, 2^{8}, 2^{24}, 2) \Rightarrow \varepsilon^\tau = 2^{-128.0}$
  Fast: $(32, 26, 2^{8}, 2^{24}, 1) \Rightarrow \varepsilon^\tau = 2^{-130.0}$

Name Protocol	Year	Instance 1	Instance 2	Proved statement
[Ste94]	1993	37.4 KB	46.1 KB	$y = Hx$, $\text{wt}(x) = w$
[Vér96]	1997	31.7 KB	38.7 KB	*message decoding*
[CVE11]	2010	-	37.4 KB	$y = Hx$, $\text{wt}(x) = w$
[AGS11]	2011	24.8 KB	-	$y = Hx$, $\text{wt}(x) = w$
[GPS21] (short)	2021	-	15.2 KB	$y = Hx$, $\text{wt}(x) = w$
[GPS21] (fast)	2021	-	19.9 KB	$y = Hx$, $\text{wt}(x) = w$
[FJR21] (short)	2021	12.9 KB	15.6 KB	$y = Hx$, $\text{wt}(x) = w$
[FJR21] (fast)	2021	20.0 KB	24.7 KB	$y = Hx$, $\text{wt}(x) = w$
Our scheme (short)	2022	9.7 KB	6.9 KB	$y = Hx$, $\text{wt}(x) \leq w$
Our scheme (fast)	2022	14.4 KB	9.7 KB	$y = Hx$, $\text{wt}(x) \leq w$

Table 2: Comparison of our protocol with state-of-the-art zero-knowledge protocols for syndrome decoding. The formulae for the communication costs of the different protocols and the used parameters are detailed in Appendix B.

We can remark that all the previous protocols prove an equality for the Hamming weight by relying on isometries (*i.e.* permutations if $\mathbb{F}_{\text{SD}} = \mathbb{F}_2$). On our side, we only prove the inequality $\text{wt}(w) \leq w$. We stress that both versions (equality or inequality) can be merely equivalent for some SD parameters. Indeed, if w is chosen sufficiently below the Gilbert-Varshamov bound and if we know there exists an SD solution x of Hamming weight w, then proving the knowledge of a solution x' with $\text{wt}(x') \leq w$ amounts to proving the knowledge of x with overwhelming probability.

4 The Signature Scheme

A signature scheme is a triplet of PPT algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$. On input 1^λ for security level λ, KeyGen outputs a pair (pk, sk) where $pk \in \{0,1\}^{\mathsf{poly}(\lambda)}$ is a public key and $sk \in \{0,1\}^{\mathsf{poly}(\lambda)}$ is a private key (a.k.a. secret key). On input a secret key sk and a message $m \in \{0,1\}^*$, Sign produces a signature $s \in \{0,1\}^{\mathsf{poly}(\lambda)}$. Verif is a deterministic algorithm which, on input a public key pk, a signature s and a message m, outputs 1 if s is a valid signature for m under pk (meaning that it is a possible output

$s \leftarrow \mathsf{Sign}(sk, m)$ for the corresponding sk) and it outputs 0 otherwise. The standard security property for a signature scheme is the *existential unforgeability* against chosen message attacks: an adversary $\mathcal{A}$ given pk and a oracle access to $\mathsf{Sign}(sk, \cdot)$ should not be able to produce a pair (s, m) satisfying $\mathsf{Verif}(pk, s, m) = 1$ (for a message m which was not queried to the signing oracle).

In this section, we show how to turn our 5-round HVZK protocol into a signature scheme using the Fiat-Shamir transform [FS87,AABN02]. After explaining the transformation, we give the description of the signature scheme and then provide a security proof in the random oracle model (ROM).

4.1 Transformation into a Non-Interactive Scheme

To transform our protocol into a non-interactive scheme, we apply the multi-round variant of the Fiat-Shamir transform [FS87] (see *e.g.* [EDV+12,CHR+16]). Concretely, we compute the challenge CH_1 and CH_2 as

$$\begin{aligned} h_1 &= \text{Hash}_1(m, \mathsf{salt}, h) \\ \text{CH}_1 &\leftarrow \text{PRG}(h_1) \end{aligned}$$

and

$$\begin{aligned} h_2 &= \text{Hash}_2(m, \mathsf{salt}, h, h') \\ \text{CH}_2 &\leftarrow \text{PRG}(h_2) \end{aligned}$$

where m is the input message, where Hash_1 and Hash_2 are some hash functions (that shall be modeled as random oracles) and where h and h' are the Round 1 and Round 3 hash commitments merged for the τ repetitions. We introduce a value salt called *salt* which is sampled from $\{0,1\}^{2\lambda}$ at the beginning of the signing process. This value is then used for each commitment to the parties' states. Without it, the security of the signature would be at most $2^{\lambda/2}$ because of the seed collisions between several signatures. Moreover, since the signature security relies on the random oracle model, we can safely replace the commitment scheme Com of the Protocol 2 by a single hash function Hash_0.

The security of the obtained scheme is lower than the soundness error of Protocol 2. Indeed, in [KZ20a], Kales and Zaverucha describe a forgery attack against signature schemes obtained by applying the Fiat-Shamir transform to 5-round protocols. Adapting this attack to our context yields a forgery cost of

$$\text{cost}_{\text{forge}} := \min_{\tau_1, \tau_2 : \tau_1 + \tau_2 = \tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau - i}} + N^{\tau_2} \right\} \tag{4}$$

with p defined in Equation (3). This is substantially lower than the target forgery cost of $1/\varepsilon$, for ε being the soundness error of Protocol 2 (see Theorem 4). We therefore need to adapt the parameters to fill this gap.

4.2 Description of the Signature Scheme

In our signature scheme, the key generation algorithm randomly samples a syndrome decoding instance (H, y) of the syndrome decoding problem with solution x (*i.e.* $y = Hx$) with security parameter λ. In order to make the key pair compact, the matrix H is pseudorandomly generated from a λ-bit seed. Specifically, a call to the KeyGen algorithm outputs a pair $(pk, sk) := \big((\mathsf{seed}_H, y), \mathsf{mseed}\big)$ generated as follows:

1. $\mathsf{mseed} \leftarrow \{0,1\}^\lambda$
2. $(\mathsf{seed}_H, x) \leftarrow \text{PRG}(\mathsf{mseed})$ where x is sampled in $\{x \in \mathbb{F}_2^m \mid \text{wt}(x) = w\}$
3. $H \leftarrow \text{PRG}(\mathsf{seed}_H)$
4. $y = Hx$; $pk = (\mathsf{seed}_H, y)$; $sk = \mathsf{mseed}$

For the sake of simplicity, we omit the re-generation of H and x from the seeds in the algorithms below and assume $pk = (H, y)$ and $sk = (H, y, x)$.

Given a secret key $sk = (H, y, x)$ and a message $m \in \{0,1\}^*$, the algorithm Sign proceeds as described in Figure 1. And given a public key $pk = (H, y)$, a signature σ and a message $m \in \{0,1\}^*$, the algorithm Verif proceeds as described in Figure 2. For the sake of clarity, as for the protocol description in Section 3.3, we use the bold notation to represent a tuple of d polynomials or of d points.

Inputs: A secret key $sk = (H, y, x)$ and a message $m \in \{0,1\}^*$.

Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.

Phase 1.0: Building of the proof witness. For all chunk $j \in [d]$,

1. Compute $Q_j(X) = \prod_{\ell \in E}(X - \gamma_\ell) \in \mathbb{F}_{\text{poly}}[X]$ where $E = \{\ell : (x_j)_\ell \neq 0\}$.
2. Compute $S_j(X) \in \mathbb{F}_{\text{poly}}[X]$ by interpolation *s.t.* $\deg S_j \leq \frac{m}{d} - 1$ and $\forall \ell \in [\frac{m}{d}], S_j(\gamma_\ell) = (x_j)_\ell$.
3. Compute $P_j(X) = S_j(X)Q_j(X)/F(X) \in \mathbb{F}_{\text{poly}}[X]$.

Phase 1.1: Preparation of the MPC-in-the-Head inputs. For each iteration $e \in [\tau]$,

1. Sample a root seed: $\mathsf{seed}^{[e]} \xleftarrow{\$} \{0,1\}^\lambda$.
2. Compute parties' seeds $\mathsf{seed}_1^{[e]}, \ldots, \mathsf{seed}_N^{[e]}$ with $\text{TreePRG}(\mathsf{salt}, \mathsf{seed})$.
3. For each party $i \in \{1, \ldots, N\}$,
   - $[\![\boldsymbol{a}_j^{[e]}]\!]_i, [\![\boldsymbol{b}_j^{[e]}]\!]_i \leftarrow \text{PRG}(\mathsf{salt}, \mathsf{seed}_i^{[e]})$, for each $j \in [t]$
   - If $i \neq N$,
     - $\{[\![c_j^{[e]}]\!]_i\}_{j\in[t]}, [\![x_A^{[e]}]\!]_i, [\![\boldsymbol{Q}^{[e]}]\!]_i, [\![\boldsymbol{P}^{[e]}]\!]_i \leftarrow \text{PRG}(\mathsf{salt}, \mathsf{seed}_i^{[e]})$
     - $\mathsf{state}_i^{[e]} = \mathsf{seed}_i^{[e]}$
   - Else,
     - $[\![x_A^{[e]}]\!]_N = x_A - \sum_{j\neq N} [\![x_A^{[e]}]\!]_j$
     - $[\![\boldsymbol{Q}^{[e]}]\!]_N = \boldsymbol{Q} - \sum_{\ell \neq N} [\![\boldsymbol{Q}^{[e]}]\!]_\ell$.
     - $[\![\boldsymbol{P}^{[e]}]\!]_N = \boldsymbol{P} - \sum_{\ell \neq N} [\![\boldsymbol{P}^{[e]}]\!]_\ell$.
     - $[\![c_j^{[e]}]\!]_N = \langle \boldsymbol{a}_j^{[e]}, \boldsymbol{b}_j^{[e]} \rangle - \sum_{\ell \neq N} [\![c_j^{[e]}]\!]_\ell$, for each $j \in [t]$
     - $\mathsf{aux}^{[e]} = ([\![x_A^{[e]}]\!]_N, [\![\boldsymbol{Q}^{[e]}]\!]_N, [\![\boldsymbol{P}^{[e]}]\!]_N, \{[\![c_j^{[e]}]\!]_N\}_{j\in[t]})$
     - $\mathsf{state}_N^{[e]} = \mathsf{seed}_N^{[e]} \mid\mid \mathsf{aux}^{[e]}$
   - Compute $\mathsf{com}_i^{[e]} = \text{Hash}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{[e]})$.

Phase 2: First challenge (randomness for the MPC protocol).

1. Compute $h_1 = \text{Hash}_1(m, \mathsf{salt}, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_{N-1}^{[\tau]}, \mathsf{com}_N^{[\tau]})$.
2. Extend hash $\{r_j^{[e]}, \boldsymbol{\varepsilon}_j^{[e]}\}_{e\in[\tau], j\in[t]} \leftarrow \text{PRG}(h_1)$ where $r_j^{[e]} \in \mathbb{F}_{\text{points}}$ and $\boldsymbol{\varepsilon_j}^{[e]} \in \mathbb{F}_{\text{points}}^d$.

Phase 3: Simulation of the MPC protocol. For each iteration $e \in [\tau]$,

1. The parties locally set $[\![x_B^{[e]}]\!] = y - H'[\![x_A^{[e]}]\!]$.
2. Then for all $j \in [t]$,
   - The parties locally compute $[\![\boldsymbol{S}^{[e]}]\!]$ by interpolation using $[\![x^{[e]}]\!] := ([\![x_A^{[e]}]\!] \mid [\![x_B^{[e]}]\!])$.
   - They locally compute $[\![\boldsymbol{S}^{[e]}(r_j^{[e]})]\!]$, $[\![\boldsymbol{Q}^{[e]}(r_j^{[e]})]\!]$ and $[\![\boldsymbol{P}^{[e]}(r_j^{[e]})]\!]$.
   - They locally set $[\![\boldsymbol{\alpha}_j^{[e]}]\!] = \boldsymbol{\varepsilon}_j^{[e]} \circ [\![\boldsymbol{Q}^{[e]}(r_j^{[e]})]\!] + [\![\boldsymbol{a}_j^{[e]}]\!]$.
   - They locally set $[\![\boldsymbol{\beta}_j^{[e]}]\!] = [\![\boldsymbol{S}^{[e]}(r_j^{[e]})]\!] + [\![\boldsymbol{b}_j^{[e]}]\!]$.
   - The parties open $[\![\boldsymbol{\alpha}_j^{[e]}]\!]$ and $[\![\boldsymbol{\beta}_j^{[e]}]\!]$ to get $\boldsymbol{\alpha}_j^{[e]}$ and $\boldsymbol{\beta}_j^{[e]}$.
   - The parties locally set

$$[\![v_j^{[e]}]\!] = -[\![c_j^{[e]}]\!] + \langle \boldsymbol{\varepsilon}_j^{[e]}, F(r_j^{[e]}) \cdot [\![\boldsymbol{P}^{[e]}(r_j^{[e]})]\!] \rangle + \langle \boldsymbol{\alpha}_j^{[e]}, [\![\boldsymbol{b}_j^{[e]}]\!] \rangle + \langle \boldsymbol{\beta}_j^{[e]}, [\![\boldsymbol{a}_j^{[e]}]\!] \rangle - \langle \boldsymbol{\alpha}_j^{[e]}, \boldsymbol{\beta}_j^{[e]} \rangle \ .$$

Phase 4: Second challenge (parties to be opened).

1. Compute $h_2 = \text{Hash}_2(m, \mathsf{salt}, h_1, \{[\![\boldsymbol{\alpha}_j^{[e]}]\!], [\![\boldsymbol{\beta}_j^{[e]}]\!], [\![v_j^{[e]}]\!]\}_{j\in[t], e\in[\tau]})$.
2. Expand hash $\{i^{*[e]}\}_{e\in[\tau]} \leftarrow \text{PRG}(h_2)$ where $i^{*[e]} \in [N]$.

Phase 5: Building of the signature. Output the signature σ built as

$$\mathsf{salt} \mid h_1 \mid h_2 \mid \left((\mathsf{state}_i^{[e]})_{i\neq i^{*[e]}} \mid \mathsf{com}_{i^{*[e]}}^{[e]} \mid \{[\![\boldsymbol{\alpha}_j^{[e]}]\!]_{i^{*[e]}}\}_{j\in[t]} \mid \{[\![\boldsymbol{\beta}_j^{[e]}]\!]_{i^{*[e]}}\}_{j\in[t]} \right)_{e\in[\tau]} \ .$$

Fig. 1: Code-based signature scheme - Signing algorithm.

4.3 Signature Properties

We now state the security of our signature scheme in the following theorem. The proof is provided in Appendix G.

Theorem 5. *Suppose the PRG used is (t, ε_{PRG})-secure and any adversary running in time t has at most an advantage ε_{SD} against the underlying d-split syndrome decoding problem. Model* Hash_0, Hash_1 *and* Hash_2 *as*

Inputs: A public key $pk = (H, y)$, a signature σ and a message $m \in \{0,1\}^*$.

1. Parse the signature σ as

$$\mathsf{salt} \mid h_1 \mid h_2 \mid \left((\mathsf{state}_i^{[e]})_{i \neq i^{*[e]}} \mid \mathsf{com}_{i^{*[e]}}^{[e]} \mid \{[\![\boldsymbol{\alpha}^{[\boldsymbol{e}]}{}_j]\!]_{i^{*[e]}}\}_{j \in [t]} \mid \{[\![\boldsymbol{\beta}^{[\boldsymbol{e}]}{}_j]\!]_{i^{*[e]}}\}_{j \in [t]} \right)_{e \in [\tau]} .$$

2. Extend hash $\{r_j^{[e]}, \boldsymbol{\epsilon}_j^{[e]}\}_{e \in [\tau], j \in [t]} \leftarrow \mathrm{PRG}(h_1)$ where $r_j^{[e]} \in \mathbb{F}_{\text{points}}$ and $\boldsymbol{\epsilon}_{\boldsymbol{j}}{}^{[e]} \in \mathbb{F}_{\text{points}}^d$.
3. Extend hash $\{i^{*[e]}\}_{e \in [\tau]} \leftarrow \mathrm{PRG}(h_2)$ where $i^{*[e]} \in [N]$.
4. For each iteration $e \in [\tau]$,
   - For each $i \neq i^{*[e]}$, computes $\mathsf{com}_i^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{[e]})$.
   - Using $\{\mathsf{state}_i^{[e]}\}_{i \neq i^{*[e]}}$, simulate all the parties except for $i^{*[e]}$ as in the Phase 3 of the signing algorithm and get $[\![\boldsymbol{\alpha}_1]\!], \ldots, [\![\boldsymbol{\alpha}_t]\!], [\![\boldsymbol{\beta}_1]\!], \ldots, [\![\boldsymbol{\beta}_t]\!], [\![v]\!]$ for all parties except for $i^{*[e]}$.
   - Compute $[\![v_j^{[e]}]\!]_{i^{*[e]}} := -\sum_{i \neq i^{*[e]}} [\![v_j^{[e]}]\!]_i$ for all $j \in [t]$.
5. Compute $h_1' = \mathrm{Hash}_1(m, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_{N-1}^{[\tau]}, \mathsf{com}_N^{[\tau]})$.
6. Compute $h_2' = \mathrm{Hash}_2(m, \{[\![\boldsymbol{\alpha}_j^{[e]}]\!], [\![\boldsymbol{\beta}_j^{[e]}]\!], [\![v_j^{[e]}]\!]\}_{j \in [t], e \in [\tau]})$.
7. Output Accept iff $h_1' \stackrel{?}{=} h_1$ and $h_2' \stackrel{?}{=} h_2$.

Fig. 2: Code-based signature scheme - Verification algorithm.

random oracles where Hash_0, Hash_1 *and* Hash_2 *have* 2λ*-bit output length. Then chosen-message adversary against the signature scheme depicted in Figure 1, running in time* t*, making* q_s *signing queries, and making* q_0*,* q_1*,* q_2 *queries, respectively, to the random oracles, succeeds in outputting a valid forgery with probability*

$$\Pr[\mathsf{Forge}] \leq \frac{(q_0 + \tau N q_s)^2}{2 \cdot 2^{2\lambda}} + \frac{q_s(q_s + q_0 + q_1 + q_2)}{2^{2\lambda}} + q_s \cdot \tau \cdot \varepsilon_{PRG} + \varepsilon_{SD} + q_2 \cdot \varepsilon^{\tau},$$

where $\varepsilon = p + \frac{1}{N} - p \cdot \frac{1}{N}$ *and* p *defined in Equation* (3)*.*

4.4 Parameters

In what follows, we propose three parameter sets which achieve a security level of 128 bits for the signature:

- the first one shall rely on the hardness to solve the SD problem on $\mathbb{F}_2$;
- the second one shall also rely on the hardness to solve the SD problem on $\mathbb{F}_2$, but we shall use a d-split version to get polynomials over a chosen field, concretely $\mathbb{F}_{256}$;
- the last one shall rely on the hardness to solve the SD problem on $\mathbb{F}_{256}$.

Choice of the SD parameters. Let us first describe how we estimate the security level of a syndrome decoding instance for a random linear code over $\mathbb{F}_2$. The best *practical* attack for our parameters is the algorithm of May, Meurer and Thomae [MMT11]. As argued in [FJR21], we can lower bound the cost of this attack by only considering the cost of its topmost recursion step:

$$\frac{\binom{m}{w}}{\binom{k+\ell}{p}\binom{m-k-\ell}{w-p}} \cdot \left(L + \frac{L^2}{2^{\ell-p}} \right) \quad \text{with} \quad L := \frac{\binom{k+\ell}{p/2}}{2^p}.$$

As usual in ISD algorithm we need to optimize for the parameters ℓ (a number of rows) and p (a partial Hamming weight). Since we only account for the cost of the topmost level in the algorithm, this yields a slightly conservative estimate for the security level. We use this estimate to choose the parameters of our scheme.

Given these considerations, we suggest the following concrete parameters:

- Variant 1: standard binary syndrome decoding problem. We propose the parameters

$$(q, m, k, w, d) = (2, 1280, 640, 132, 1)$$

which achieve a security level of 128 bits according to the above formula.

– Variant 2: d-split binary syndrome syndrome decoding problem, where d is taken to have $m/d \leq 256$ so that $\mathbb{F}_{\text{poly}} = \mathbb{F}_{256}$. We propose the parameters

$$(q, m, k, w, d) = (2, 1536, 888, 120, 6)$$

which achieve a security of 129 bits. Indeed, the standard SD problem with the same parameters (but $d = 1$) has a security of 145 bits and we know, thanks to the Theorem 8, that there is a security loss of at most 16 bits while switching to $d = 6$.

– Variant 3: syndrome decoding instance defined over $\mathbb{F}_{256}$. The cryptanalysis of the syndrome decoding problem on a field which is larger than $\mathbb{F}_2$ has been less studied. Previous articles [CVE11,GPS21] propose parameters sets for syndrome decoding instances over $\mathbb{F}_{2^8}$ where the code length m is between 200 and 210. In our case, we choose $m = 256$ in such a way that the polynomial degree is equal to the field size. Besides being more conservative, this choice has the advantage of easing the use of a Fast Fourier Transform. Here, we estimate that the below parameter set is a conservative choice.[4] We propose the following parameters for this variant:

$$(q, m, k, w, d) = (256, 256, 128, 80, 1) \ .$$

Choice of the MPC parameters. For each variant, we suggest in Table 3 a parameter set for the MPC protocol.

Scheme	SD Parameters					MPC Parameters			
	q	m	k	w	d	$\lvert\mathbb{F}_{\text{poly}}\rvert$	$\lvert\mathbb{F}_{\text{points}}\rvert$	t	p
Variant 1	2	1280	640	132	1	2^{11}	2^{22}	6	$\approx 2^{-69}$
Variant 2	2	1536	888	120	6	2^{8}	2^{24}	5	$\approx 2^{-79}$
Variant 3	2^8	256	128	80	1	2^{8}	2^{24}	5	$\approx 2^{-78}$

Table 3: SD and MPC parameters.

To have a short signature, we take the smallest possible field $\mathbb{F}_{\text{poly}}$ since a signature transcript includes polynomials on that field. As explained in Section 3, $\mathbb{F}_{\text{poly}}$ must be a field extension of $\mathbb{F}_{\text{SD}}$ which verifies the relation $|\mathbb{F}_{\text{poly}}| \geq m/d$. Then, it remains to choose $|\mathbb{F}_{\text{points}}|$ and t. These parameters are chosen to make the false positive probability p is negligible compared to $1/N$ so that the optimal forgery strategy of an attacker is to take $\tau_1 = 1$ in the Equation (4). As a result, we just need to increase the number of iterations τ by one compared to the interactive protocol.

4.5 Implementation and Performances

For each repetition in the computation of each party, d polynomial interpolations are involved. Indeed, from $[\![x]\!]$, the parties must compute

$$[\![S_\ell]\!](X) = \sum_{i=1}^{m/d} [\![x_{\frac{m}{d}\ell+i}]\!] \cdot \prod_{j=1, j\neq i}^{m/d} \frac{X - w_j}{w_i - w_j}$$

for all $\ell \in [d]$. Then, the parties must evaluate $[\![S_\ell]\!]$ in t random evaluation points sampled by the verifier, for all $\ell \in [d]$. The natural way to implement that is to compute the coefficients of all the polynomials $\{[\![S_\ell]\!]\}_\ell$

[4] More cryptanalysis of the SD problem over $\mathbb{F}_{256}$ would be welcome to get more confidence in the choice of the parameters. Such research is out of the scope of present article.

from $[\![x]\!]$, then to evaluate these polynomials t times. However this implies that the signer must realize $\tau \cdot N \cdot d$ interpolations. Instead, the signer can compute the vector $u(r)$ defined as

$$u(r) = \left(\prod_{j=1, j\neq i}^{m/d} \frac{r - w_j}{w_i - w_j} \right)_{1 \leq i \leq \frac{m}{d}}$$

for each evaluation point r, and then use these vectors in the computation of all the parties as

$$[\![S_\ell(r)]\!] = \langle [\![x_\ell]\!], u(r) \rangle$$

where $[\![x_\ell]\!]$ is the ℓ^{th} chunk of $[\![x]\!]$. By proceeding this way, the number of (transposed) interpolations done by the signer is of $\tau \cdot t$.

To reduce the computational cost of the interpolations, we can make use of a Fast Fourier Transform (FFT). We are working on field extensions of $\mathbb{F}_2$, so we can use the Additive FFT independently introduced by Wang-Zhu in 1988 [WZ88] and by Cantor in 1989 [Can89], which was further improved in [vzGG03,GM10]. Although such additive FFT exists for any extension of $\mathbb{F}_2$, the algorithms are simpler for a field of size $2^{(2^i)}$ for some i, which is why we define $\mathbb{F}_{\text{poly}}$ as $\mathbb{F}_{256}$. On such a field $\mathbb{F}$, we indeed have an efficient additive FFT using $\frac{1}{2}|\mathbb{F}| \log_2 |\mathbb{F}|$ multiplications to evaluate a polynomial (of degree lower than $|\mathbb{F}|$) in $|\mathbb{F}|$ points.

We implemented the signature scheme in C. In our implementation, the pseudo-randomness is generated using AES in counter mode and the hash function is instantiated with SHAKE. We benchmarked our scheme on a 3.8 GHz Intel Core i7 CPU with support of AVX2 and AES instructions. All the reported timings were measured on this CPU while disabling Intel Turbo Boost.

Remark 2. Another motivation for using $\mathbb{F}_{\text{poly}} = \mathbb{F}_{256}$ is that some Intel processors have dedicated instructions for $\mathbb{F}_{256}$ arithmetic. We therefore expect substantial speed-ups for the instances of our signature scheme using $\mathbb{F}_{\text{poly}} = \mathbb{F}_{256}$ on these processors. Optimizing and benchmarking such implementations is left for future research.

We instantiate two trade-offs per variant: the first one lowering communication cost to produce short signatures, and the second one lowering computational cost to get a fast signature computation. We obtain the parameters and sizes described in Table 4. We provide the measured computational performances of our signature implementation in Table 5.

λ	Scheme	Aim	Parameters		$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{sk}\rvert$	Signature	
			N	τ			$\lvert\mathsf{sgn}\rvert$ (max)	$\lvert\mathsf{sgn}\rvert$ (avg, std)
128	Variant 1	Fast	32	27	96	16	16 422	16 006, 446
128		Short	256	17	96	16	11 193	11 160, 127
128	Variant 2	Fast	32	27	97	16	17 866	17 406, 494
128		Short	256	17	97	16	12 102	12 066, 141
128	Variant 3	Fast	32	27	144	16	12 115	11 835, 302
128		Short	256	17	144	16	8 481	8 459, 86

Table 4: Parameters (N, τ) with the achieved communication costs (in bytes).

Future investigations. We tried to optimize the implementation using some algorithmetic tricks, but we did not yet investigate the possible software optimizations like vectorialization or bitslicing. Although the variants 1 and 2 are more conservative because they rely on the hardness of the binary syndrome decoding problem, variant 3 is more promising in terms of signature size and computation time. While we have investigated parameter sets where $\mathbb{F}_{\text{SD}}$ is a field extension of $\mathbb{F}_2$, more cryptanalysis for the SD problem on those fields

λ	Variant	Aim	Keygen	Sign	Verify
128	Variant 1	Fast	n/a†		
128		Short			
128	Variant 2	Fast	0.03 ms 114 162 cycles	13.4 ms 52 463 114 cycles	12.7 ms 50 306 845 cycles
128		Short	0.03 ms 113 852 cycles	64.2 ms 251 099 099 cycles	60.7 ms 243 055 474 cycles
128	Variant 3	Fast	0.01 ms 49 181 cycles	6.4 ms 25 253 580 cycles	5.9 ms 23 816 143 cycles
128		Short	0.01 ms 49 057 cycles	29.5 ms 114 226 505 cycles	27.1 ms 108 541 768 cycles

Table 5: Benchmarks of our signature implementation. Timings are averaged over 10 000 measurements. The CPU clock cycles have been measured using SUPERCOP (`https://bench.cr.yp.to/supercop.html`).
† *We only have a proof of concept implementation with irrelevant timings.*

as well as on non-binary fields would be welcome. An interesting idea would be to instantiate our scheme with a prime field $\mathbb{F}_{\text{SD}}$ for which the Number-Theoretic Transform (NTT) is defined. If $\mathbb{F}_{\text{SD}}$ is large enough, we could then take the same field for $\mathbb{F}_{\text{poly}}$ than $\mathbb{F}_{\text{SD}}$ and we would have fast polynomial interpolations and simpler multiplication operations.

5 Comparison

In this section, we compare our scheme to different code-based and post-quantum signature schemes from the literature.

5.1 Comparison with Other Code-Based Signature Schemes

In the state of the art, there exist two approaches to build signatures. On one hand, there is the hash-and-sign paradigm which relies on the existence of a (code-based) trapdoor permutation. Wave [DST19] is a code-based signature scheme in this paradigm. Such schemes are often more vulnerable to structural attacks. On the other hand, further signature schemes are constructed by applying the Fiat-Shamir transform to (zero-knowledge) identification schemes, which can rely on weaker assumptions (and typically the SD for random linear codes). Historically such schemes, like the famous Stern protocol, give rise to large signatures because of the high soundness error of the underlying identification scheme (2/3 or 1/2). To avoid this issue, a solution consists in relying on different code-based problems. For instance, LESS is a recent scheme which security relies on the hardness of the Linear Code Equivalence problem [BMPS20,BBPS21]. Another direction is to find a way to adapt the Schnorr-Lyubashevsky approach to code-based cryptography. Durandal is a recent scheme following this approach [ABG+19]. More recently, some works [GPS21,FJR21,BGKM22] have obtained better soundness by relying on the MPC-in-the-Head principle. The proposed schemes achieve small signature sizes at the cost of slower computation. Depending on the setting, they can produce signatures with different trade-offs between the signature size and the computational cost. The current work follows this approach while achieving better trade-offs than any of these previous works.

Table 6 compares the performances of our scheme with the current code-based signature state of the art, for the 128-bit security level.[5] We observe that our scheme outperforms all the existing code-based signatures for the $|\mathsf{sgn}|+|\mathsf{pk}|$ metric. Depending on the parameters, it can even produce signatures such that $|\mathsf{sgn}|+|\mathsf{pk}|$ is below the symbolic cap of 10 KB. Regardless of the key size, Wave still achieves the shortest signatures.

[5] We did not include "Sig 3" from [BGKM22] since it is similar to [FJR21] with slight differences (*message decoding* setting) which do not improve the scheme.

Scheme Name	Year	$\|\mathsf{sgn}\|$	$\|\mathsf{pk}\|$	t_{sgn}	t_{verif}	Assumption
Wave	2019	2.07 K	3.2 M	300	-	SD over $\mathbb{F}_3$ (large weight) $(U, U+V)$-codes indisting.
Durandal - I	2018	3.97 K	14.9 K	4	5	Rank SD over $\mathbb{F}_{2^m}$
Durandal - II	2018	4.90 K	18.2 K	5	6	Rank SD over $\mathbb{F}_{2^m}$
LESS-FM - I	2020	9.77 K	15.2 K	-	-	Linear Code Equivalence
LESS-FM - II	2020	206 K	5.25 K	-	-	Perm. Code Equivalence
LESS-FM - III	2020	11.57 K	10.39 K	-	-	Perm. Code Equivalence
[GPS21]-256	2021	24.0 K	0.11 K	-	-	SD over $\mathbb{F}_{256}$
[GPS21]-1024	2021	19.8 K	0.12 K	-	-	SD over $\mathbb{F}_{1024}$
[FJR21] (fast)	2021	22.6 K	0.09 K	13	12	SD over $\mathbb{F}_2$
[FJR21] (short)	2021	16.0 K	0.09 K	62	57	SD over $\mathbb{F}_2$
[BGKM22] - Sig1	2022	23.7 K	0.1 K	-	-	SD over $\mathbb{F}_2$
[BGKM22] - Sig2	2022	20.6 K	0.2 K	-	-	(QC)SD over $\mathbb{F}_2$
Our scheme - Var1f	2022	15.6 K	0.09 K	-	-	SD over $\mathbb{F}_2$
Our scheme - Var1s	2022	10.9 K	0.09 K	-	-	SD over $\mathbb{F}_2$
Our scheme - Var2f	2022	17.0 K	0.09 K	13	13	SD over $\mathbb{F}_2$
Our scheme - Var2s	2022	11.8 K	0.09 K	64	61	SD over $\mathbb{F}_2$
Our scheme - Var3f	2022	11.5 K	0.14 K	6	6	SD over $\mathbb{F}_{256}$
Our scheme - Var3s	2022	8.26 K	0.14 K	30	27	SD over $\mathbb{F}_{256}$

Table 6: Comparison of our scheme with signatures from the literature (128-bit security). The sizes are in bytes and the timings are in milliseconds. Reported timings are from the original publications: Wave has been benchmarked on a 3.5 Ghz Intel Xeon E3-1240 v5, Durandal on a 2.8 Ghz Intel Core i5-7440HQ, while [FJR21] and our scheme on a 3.8 GHz Intel Core i7.

In terms of security, our scheme has the advantage of relying on the hardness of one of the oldest problems of the code-based cryptography, namely the syndrome decoding for random linear codes in Hamming weight metric.

5.2 Comparison with other Post-Quantum Signature Schemes

Finally, we compare in Table 7 our construction with other signature schemes aiming at post-quantum security. All these schemes have very short public keys and secret keys (less than 100 bytes for 128-bit security), thus it is not a point for comparison. Depending on the chosen parameters, our scheme can be competitive with Picnic3 [KZ20b] and the recently proposed "Picnic4" [KZ21] which also rely on the MPC-in-the-Head paradigm. Like Picnic4, it can produce signatures with a size of around 8 KB. However, our scheme is arguably more conservative in terms of security since Picnic is based on the hardness of inverting LowMC [ARS+15], a cipher with unconventional design choices, while our scheme is based on the hardness of the syndrome decoding problem on linear codes, which has a long cryptanalysis history and is believed to be very robust. Banquet [BdK+21] is a signature scheme for which the security is based on the hardness of inverting AES (instead of LowMC), which can also be argued to be a conservative choice. Our scheme over $\mathbb{F}_2$ is competitive with Banquet: slightly shorter and slightly slower (but the timing could be optimized). On the other hand, our scheme on $\mathbb{F}_{256}$ clearly outperforms Banquet. Our scheme can also be competitive with SPHINCS$^+$ [BHK+19] depending on the exact criteria. For similar signature sizes, our signature computation is significantly faster while our signature verification is significantly slower than those of SPHINCS$^+$.

References

AABN02. Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 418–433. Springer, Heidelberg, April / May 2002.

Scheme Name	$\lvert\mathsf{sgn}\rvert$	t_{sgn}	t_{verif}
SPHINCS$^+$-128f	16.7 K	14	1.7
SPHINCS$^+$-128s	7.7 K	239	0.7
Picnic3	12.3 K	5.2	4.0
Picnic4	7.8 K	≈ 20	≈ 20
Banquet (fast)	19.3 K	6	5
Banquet (short)	13.0 K	44	40
Our scheme - Var1f	15.6 K	-	-
Our scheme - Var1s	10.9 K	-	-
Our scheme - Var2f	17.0 K	13	13
Our scheme - Var2s	11.8 K	64	61
Our scheme - Var3f	11.8 K	6	6
Our scheme - Var3s	8.3 K	30	27

Table 7: Comparison of our scheme with signatures from the literature. The sizes are in bytes and the timings are in milliseconds. The benchmarks of the other schemes have been realized on a Intel Xeon W-2133 CPU at 3.60GHz, the values for SPHINCS$^+$ and Banquet have been extracted from [BdK+21] while the values for Picnic3 have been extracted from its original publication [KZ20b].

ABG+19. Nicolas Aragon, Olivier Blazy, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. Durandal: A rank metric based signature scheme. In *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 728–758. Springer, Heidelberg, May 2019.

ACBH13. Sidi Mohamed El Yousfi Alaoui, Pierre-Louis Cayrel, Rachid El Bansarkhani, and Gerhard Hoffmann. Code-Based Identification and Signature Schemes in Software. In *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings*, volume 8128 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2013.

AGS11. Carlos Aguilar, Philippe Gaborit, and Julien Schrek. A new zero-knowledge code based identification scheme with reduced communication. In *2011 IEEE Information Theory Workshop*, pages 648–652, 2011.

ARS+15. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, April 2015.

BBC+19. Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. A Finite Regime Analysis of Information Set Decoding Algorithms. *Algorithms*, 12(10):209, 2019.

BBPS21. Alessandro Barenghi, Jean-Francois Biasse, Edoardo Persichetti, and Paolo Santini. LESS-FM: Fine-tuning Signatures from the Code Equivalence Problem. International Workshop on Post-Quantum Cryptography (PQCrypto), 2021.

BdK+21. Carsten Baum, Cyprien de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 266–297. Springer, Heidelberg, May 2021.

Bea92. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

BGKM22. Loïc Bidoux, Philippe Gaborit, Mukul Kulkarni, and Victor Mateu. Code-based Signatures from New Proofs of Knowledge for the Syndrome Decoding Problem, 2022.

BHK+19. Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ signature framework. In *ACM CCS 2019*, pages 2129–2146. ACM Press, November 2019.

BMPS20. Jean-François Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. LESS is more: Code-based signatures without syndromes. In *AFRICACRYPT 20*, volume 12174 of *LNCS*, pages 45–65. Springer, Heidelberg, July 2020.

BN20. Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 495–526. Springer, Heidelberg, May 2020.

Can89. David G. Cantor. On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A*, 50:285–300, 1989.

CDG+. Melissa Chase, David Derler, Steven Goldfeder, Jonathan Katz, Vladimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha. The Picnic Signature Scheme – Design Document. Version 2.2 – 14 April 2020.

CHR+16. Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. From 5-pass MQ-based identification to MQ-based signatures. In *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 135–165. Springer, Heidelberg, December 2016.

CVE11. Pierre-Louis Cayrel, Pascal Véron, and Sidi Mohamed El Yousfi Alaoui. A zero-knowledge identification scheme based on the q-ary syndrome decoding problem. In *SAC 2010*, volume 6544 of *LNCS*, pages 171–186. Springer, Heidelberg, August 2011.

DST19. Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. Wave: A new family of trapdoor one-way preimage sampleable functions based on codes. In *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 21–51. Springer, Heidelberg, December 2019.

EDV+12. Sidi Mohamed El Yousfi Alaoui, Özgür Dagdelen, Pascal Véron, David Galindo, and Pierre-Louis Cayrel. Extended security arguments for signature schemes. In *AFRICACRYPT 12*, volume 7374 of *LNCS*, pages 19–34. Springer, Heidelberg, July 2012.

FJR21. Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Shared Permutation for Syndrome Decoding: New Zero-Knowledge Protocol and Code-Based Signature. Cryptology ePrint Archive, Report 2021/1576, 2021.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

GG07. Philippe Gaborit and Marc Girault. Lightweight code-based identification and signature. In *IEEE International Symposium on Information Theory, ISIT 2007, Nice, France, June 24-29, 2007*, pages 191–195. IEEE, 2007.

GM10. Shuhong Gao and Todd Mateer. Additive Fast Fourier Transforms Over Finite Fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, 2010.

GPS21. Shay Gueron, Edoardo Persichetti, and Paolo Santini. Designing a Practical Code-based Signature Scheme from Zero-Knowledge Proofs with Trusted Setup. Cryptology ePrint Archive, Report 2021/1020, 2021.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

KZ20a. Daniel Kales and Greg Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In *CANS 20*, volume 12579 of *LNCS*, pages 3–22. Springer, Heidelberg, December 2020.

KZ20b. Daniel Kales and Greg Zaverucha. Improving the performance of the Picnic signature scheme. *IACR TCHES*, 2020(4):154–188, 2020.

KZ21. Daniel Kales and Greg Zaverucha. Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures. Preliminary Draft, October 29, 2021, 2021.

LN17. Yehuda Lindell and Ariel Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In *ACM CCS 2017*, pages 259–276. ACM Press, October / November 2017.

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, Heidelberg, December 2011.

PS00. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

Ste94. Jacques Stern. A new identification scheme based on syndrome decoding. In *CRYPTO'93*, volume 773 of *LNCS*, pages 13–21. Springer, Heidelberg, August 1994.

TS16. Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 144–161. Springer, Heidelberg, 2016.

Vér96. Pascal Véron. Improved identification schemes based on error-correcting codes. *Appl. Algebra Eng. Commun. Comput.*, 8(1):57–69, 1996.

vzGG03. J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003.

WZ88. Y. Wang and X. Zhu. A fast algorithm for the Fourier transform over finite fields and its VLSI implementation. *IEEE Journal on Selected Areas in Communications*, 6(3):572–577, 1988.

– Supplementary Material –

A Proof of Theorem 1

Proof. To prove the theorem, we build below an algorithm $\mathcal{A}_1$ to solve the traditional SD problem of parameters (m, k, w) using an algorithm $\mathcal{A}_d$ which solves the d-split SD problem with the same parameters.

Algorithm $\mathcal{A}_1$ (on input an SD instance (H, y)):

1. Sample a permutation σ of $\{1, \ldots, m\}$.
2. Permute the columns of H using σ to get $\hat{H}$.
3. Run $\mathcal{A}_d$ on input $(\hat{H}, y)$ to get $\hat{x}$.
4. If $\hat{x} = \perp$, return $\perp$.
5. Permute the coordinates of $\hat{x}$ using σ^{-1} to get x.
6. Return x.

The probability to transform an SD instance into a d-split SD instance in Step 2 is $\binom{m/d}{w/d}^d / \binom{m}{w}$. Thus we have

$$\begin{aligned}
\varepsilon_1 &:= \Pr[\mathcal{A}_1(H, y) \neq \perp] \\
&\geq \Pr[\mathcal{A}_1(H, y) \neq \perp \cap (\hat{H}, y) \text{ is a } d\text{-split SD}] \\
&= \frac{\binom{m/d}{w/d}^d}{\binom{m}{w}} \cdot \Pr[\mathcal{A}_1(H, y) \neq \perp \mid (\hat{H}, y) \text{ is a } d\text{-split SD}] \\
&= \frac{\binom{m/d}{w/d}^d}{\binom{m}{w}} \cdot \Pr[\mathcal{A}_d(\hat{H}, y) \neq \perp \mid (\hat{H}, y) \text{ is a } d\text{-split SD}] \\
&= \frac{\binom{m/d}{w/d}^d}{\binom{m}{w}} \cdot \varepsilon_d
\end{aligned}$$

B Communication Costs of ZK PoK for Syndrome Decoding

In this section, we exhibit formulas for the communication cost of state-of-the-art zero-knowledge proof-of-knowledge of a syndrome decoding instance:

- the Stern's protocol [Ste94],
- the Véron's protocol [Vér96],
- the [CVE11]'s protocol,
- the [AGS11]'s protocol,
- the [GPS21]'s protocol,
- and the [FJR21]'s protocol.

We consider the SD problem over $\mathbb{F}_q$ with parameters (m, k, w) and a target soundness error of $2^{-\lambda}$. To achieve this soundness error, we need to execute these protocols many times. In what follows, we denote τ the number of executions.

We describe below the *mean* communication cost of all the protocols. We *exclude the challenges* from all these costs since they are of very moderate impact and do not count whenever making the protocol non-iterative. Moreover, we do not take the implementation details into consideration, meaning that the size of

all the elements of the proofs correspond to their Shannon entropy. For example, the cost of a permutation of $\{1, \ldots, m\}$ would be about $\log_2(m!)$ bits.

To ease the formulae for the communication costs, we define the following costs:

- sd is the cost to send a random seed (λ bits);
- dig is the cost to send a hash digest or a commitment output (2λ bits);
- ptx is the cost to send a code input message ($k \cdot \log_2 q$ bits);
- cod is the cost to send a codeword ($m \cdot \log_2 q$ bits);
- nse is the cost to send a noise vector ($\log_2(\binom{m}{w} \cdot (q-1)^w)$ bits);
- iso is the cost to send a isometry of $\mathbb{F}_q^m$ ($\log_2(m! \cdot (q-1)^m)$ bits).

We note that the cost iso is only used whenever the isometry to be sent cannot be pseudorandomly generated from a seed but must be computed with respect to other elements of the proof.

Stern's protocol. The soundness error of this protocol is $(2/3)^\tau$ and its optimized cost is

$$\mathsf{dig} + \tau \cdot [\mathsf{dig} + (2 \cdot \mathsf{sd} + 2 \cdot \mathsf{cod} + \mathsf{nse})/3].$$

Véron's protocol. The soundness error of this protocol is $(2/3)^\tau$ and its optimized cost is

$$\mathsf{dig} + \tau \cdot [\mathsf{dig} + (2 \cdot \mathsf{sd} + \mathsf{ptx} + \mathsf{cod} + \mathsf{nse})/3].$$

CVE10's protocol. The soundness error of this protocol is $\left(\frac{q}{2(q-1)}\right)^\tau$ and its cost is

$$\mathsf{dig} + \tau \cdot [\mathsf{dig} + \mathsf{cod} + (\mathsf{sd} + \mathsf{nse})/2].$$

AGS11's protocol ($q = 2$). The soundness error of this protocol is $\left(\frac{k+1}{2k}\right)^\tau$ and its cost is

$$2 \cdot \mathsf{dig} + \tau \cdot [\mathsf{dig} + (\mathsf{sd} + \mathsf{ptx} + \mathsf{cod} + \mathsf{nse})/2].$$

GPS21's protocol. The soundness error of this protocol is

$$\max_{M-\tau \le k \le M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot q^{k-M+\tau}} \right\}$$

and its cost is

$$2 \cdot \mathsf{dig} + \tau \cdot \log_2 \frac{M}{\tau} \cdot \mathsf{sd} + \tau \cdot [\mathsf{iso} + \mathsf{cod} + \log_2(q) \cdot \mathsf{dig}].$$

In [GPS21], the authors give another formula for the communication cost for their signature scheme. Since we are here comparing interactive protocols, we consider a 5-round variant of their scheme to have a fairer comparison. In their article, they have additional costs because they consider only the 3-round variant to build an efficient signature.

In Table 2, we take the following parameters for this protocol:

- Variant "Fast": $(M, \tau) = (512, 23)$;
- Variant "Short": $(M, \tau) = (2048, 17)$.

FJR21's protocol. The soundness error of this protocol is

$$\max_{M-\tau \le k \le M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot N^{k-M+\tau}} \right\}$$

and its cost is

$$2 \cdot \mathsf{dig} + \tau \cdot \log_2 \frac{M}{\tau} \cdot \mathsf{sd} + \tau \cdot [(\mathsf{cod} + \mathsf{ptx} + \mathsf{nse}) + \log_2(N) \cdot \mathsf{sd} + \mathsf{dig}].$$

In Table 2, we take the same parameters as in [FJR21]:

- Variant "Fast": $(N, M, \tau) = (8, 187, 49)$;
- Variant "Short": $(N, M, \tau) = (32, 389, 28)$.

C Schwartz-Zippel Lemma and Variants

Lemma 3 (Schwartz-Zippel, multi-point variant). *Let $R \in \mathbb{F}[X]$ be of degree $d > 0$; for any $\mathbb{S} \subset \mathbb{F}$ and any $t \geq 1$,*

$$\Pr_{r_1,\ldots,r_t \leftarrow \mathbb{S}}[R(r_1) = 0 \cap \ldots \cap R(r_t) = 0 \mid \{r_i\} \textit{ are distinct}] \leq \frac{\binom{d}{t}}{\binom{|\mathbb{S}|}{t}} .$$

Proof. There is $\binom{|\mathbb{S}|}{t}$ possible different draws when we uniformly sample t distinct elements in $\mathbb{S}$. But R can have at most d roots, so the studied event occurs for at most $\binom{d}{t}$ of these draws. □

Lemma 4 (Schwartz-Zippel, multi-point variant 2). *Let $R \in \mathbb{F}[X]$ be of degree $d > 0$; for any $\mathbb{S} \subset \mathbb{F}$ and any $t, \ell \geq 1$,*

$$\Pr_{r_1,\ldots,r_t \leftarrow \mathbb{S}}[\#\{i : R(r_i) = 0\} = \ell \mid \{r_i\} \textit{ are distinct}] \leq \frac{\max_{i \leq d}\left\{\binom{i}{\ell} \cdot \binom{|\mathbb{S}|-i}{t-\ell}\right\}}{\binom{|\mathbb{S}|}{t}} .$$

If $t \cdot d \leq \ell \cdot (|\mathbb{S}| - 1)$, we have

$$\Pr_{r_1,\ldots,r_t \leftarrow \mathbb{S}}[\#\{i : R(r_i) = 0\} = \ell \mid \{r_i\} \textit{ are distinct}] \leq \frac{\binom{d}{\ell} \cdot \binom{|\mathbb{S}|-d}{t-\ell}}{\binom{|\mathbb{S}|}{t}} .$$

Proof. There is $\binom{|\mathbb{S}|}{t}$ possible different draws when we uniformly sample t distinct elements in $\mathbb{S}$. But R can have at most d roots. Let us denote $i \leq d$ the number of roots of R in $\mathbb{S}$. The studied event occurs for $\binom{i}{\ell} \cdot \binom{|\mathbb{S}|-i}{t-\ell}$ possible draws. We thus get

$$\Pr_{r_1,\ldots,r_t \leftarrow \mathbb{S}}[\#\{i : R(r_i) = 0\} = \ell \mid \{r_i\} \text{ are distinct}] \leq \frac{\max_{i \leq d}\left\{\binom{i}{\ell} \cdot \binom{|\mathbb{S}|-i}{t-\ell}\right\}}{\binom{|\mathbb{S}|}{t}} .$$

Now, let us assume that $\ell \cdot (|\mathbb{S}| - 1) \geq t \cdot d$. For all i in $\{0, \ldots, d-1\}$, we have

$$(i+1) \cdot t \leq d \cdot t \leq \ell \cdot (|\mathbb{S}| - 1) .$$

which is equivalent to

$$(i+1)(|\mathbb{S}| - i - t + \ell) \geq (i + 1 - \ell)(|\mathbb{S}| - i)$$

We deduce that, when $i + 1 > \ell$,

$$\begin{aligned}\binom{i+1}{\ell} \cdot \binom{|\mathbb{S}| - (i+1)}{t-\ell} &= \frac{i+1}{i+1-\ell} \cdot \binom{i}{\ell} \cdot \frac{|\mathbb{S}| - i - (t-\ell)}{|\mathbb{S}| - i} \cdot \binom{|\mathbb{S}| - i}{t-\ell} \\ &\geq \binom{i}{\ell}\binom{|\mathbb{S}| - i}{t-\ell} .\end{aligned}$$

Thus,

$$\binom{d}{\ell}\binom{|\mathbb{S}| - d}{t-\ell} \geq \binom{d-1}{\ell}\binom{|\mathbb{S}| - (d-1)}{t-\ell} \geq \ldots \geq \binom{\ell}{\ell}\binom{|\mathbb{S}| - \ell}{t-\ell}$$

and we have, for $i < \ell$,

$$\binom{i}{\ell} \cdot \binom{s-i}{t-\ell} = 0 .$$

So,

$$\max_{i \leq d}\left\{\binom{i}{\ell} \cdot \binom{|\mathbb{S}| - i}{t-\ell}\right\} = \binom{d}{\ell}\binom{|\mathbb{S}| - d}{t-\ell} .$$

□

D Splitting Lemma

In our proofs, we shall make use of the following lemma from [PS00]:

Lemma 5 (Splitting Lemma). *Let X and Y be two finite sets, and let $A \subseteq X \times Y$ such that*

$$\Pr\left[(x,y) \in A \mid (x,y) \leftarrow X \times Y\right] \geq \varepsilon \ .$$

For any $\alpha \in [0,1)$, let

$$B = \left\{(x,y) \in X \times Y \;\middle|\; \Pr\left[(x,y') \in A \mid y' \leftarrow Y\right] \geq (1-\alpha)\cdot\varepsilon\right\} \ .$$

We have:

1. $\Pr\left[(x,y) \in B \mid (x,y) \leftarrow X \times Y\right] \geq \alpha \cdot \varepsilon$
2. $\Pr\left[(x,y) \in B \mid (x,y) \leftarrow A\right] \geq \alpha$.

E Proof of Theorem 3 (HVZK of Protocol 2)

We give hereafter the proof of Theorem 3.

Proof. We first describe an internal HVZK simulator $\mathcal{S}$ which on input a pair of challenges $(\text{Ch}_1, \text{Ch}_2) = (\{r_j, \boldsymbol{\varepsilon}_j\}_{j\in[t]},\ i^*)$, outputs the corresponding responses $(\text{Rsp}_1, \text{Rsp}_2)$, as follows:

1. Sample a root seed in $\{0,1\}^\lambda$.
2. Generate the parties' seeds $\{(\mathsf{seed}_i, \rho_i)\}_i$ with TreePRG(seed).
3. For each party $i \in \{1,\ldots,N\}\backslash\{i^*\}$,
   - $[\![\boldsymbol{a}_j]\!]_i, [\![\boldsymbol{b}_j]\!]_i \leftarrow \text{PRG}(\mathsf{seed}_i)$, for each $j \in [t]$
   - If $i \neq N$,
     - $\{[\![c_j]\!]_i\}_{j\in[t]}, [\![x_A]\!]_i, [\![\boldsymbol{Q}]\!]_i, [\![\boldsymbol{P}]\!]_i \leftarrow \text{PRG}(\mathsf{seed}_i)$
     - $\mathsf{state}_i = \mathsf{seed}_i$
   - Else,
     - $[\![x_A]\!]_N \leftarrow \{0,1\}^k$
     - $[\![\boldsymbol{Q}]\!]_N \leftarrow (\mathbb{F}_{\text{poly}_{w-1}}[X])^d$.
     - $[\![\boldsymbol{P}]\!]_N \leftarrow (\mathbb{F}_{\text{poly}_{w-1}}[X])^d$.
     - $[\![c_j]\!]_N \leftarrow \mathbb{F}_{\text{points}}$, for each $j \in [t]$
     - $\mathsf{aux} = ([\![x_A]\!]_N, [\![\boldsymbol{Q}]\!]_N, [\![\boldsymbol{P}]\!]_N, \{[\![c_j]\!]_N\}_{j\in[t]})$
     - $\mathsf{state}_N = \mathsf{seed}_N \mid\mid \mathsf{aux}$
   - Simulate the computation of the party i to get $\{[\![\boldsymbol{\alpha}_j]\!]_i, [\![\boldsymbol{\beta}_j]\!]_i, [\![v_j]\!]_i\}_{j\in[t]}$.
4. For party i^*, for each $j \in [t]$,
   - $[\![\boldsymbol{\alpha}_j]\!]_{i^*} \leftarrow \mathbb{F}^d_{\text{points}}$,
   - $[\![\boldsymbol{\beta}_j]\!]_{i^*} \leftarrow \mathbb{F}^d_{\text{points}}$,
   - $[\![v_j]\!]_{i^*} = -\sum_{i\neq i^*} [\![v_j]\!]_i$.
5. Output the responses
   - $\text{Rsp}_1 = \text{Hash}([\![\boldsymbol{\alpha}_1]\!], [\![\boldsymbol{\beta}_1]\!], [\![v_1]\!], \ldots, [\![\boldsymbol{\alpha}_t]\!], [\![\boldsymbol{\beta}_t]\!], [\![v_t]\!])$,
   - $\text{Rsp}_2 = \big((\mathsf{state}_i, \rho_i)_{i\neq i^*} \mid \mathsf{com}_{i^*} \mid \{[\![\boldsymbol{\alpha}_j]\!]_{i^*}\}_{j\in[t]} \mid \{[\![\boldsymbol{\beta}_j]\!]_{i^*}\}_{j\in[t]}\big)$.

We now show that the transcript produced by the above simulator is $(t, \varepsilon_{\text{PRG}})$-instinguishable from a real transcript of Protocol 2 with challenges $\{r_j, \boldsymbol{\varepsilon}_j\}_{j\in[t]}$ and i^*. To this aim, we describe a sequence of simulators.

Simulator 0 (real world). This simulator takes as input the challenges $\{r_j, \boldsymbol{\varepsilon}_j\}_{j\in[t]}$ and i^*, as well as the witness x_A, runs a genuine execution of Protocol 2 and output the corresponding responses. The latter are hence identically distributed as a the responses in a real-world transcript (for the given challenges).

Simulator 1. Same as Simulator 0, but uses true randomness instead of seed-derived randomness for party i^*. If $i^* = N$, the values $[\![x_A]\!]_N$, $[\![\boldsymbol{Q}]\!]_N$, $[\![\boldsymbol{P}]\!]_N$ and $\{[\![c_j]\!]_N\}_{j\in[t]}$ are computed as described in the protocol (only $[\![\boldsymbol{a}_j]\!]_N$ and $[\![\boldsymbol{b}_j]\!]_N$ are generated from true randomness).

It is easy to see that the probability of distinguishing Simulator 1 and Simulator 0 in running time t is no more than ε_{PRG}.

Simulator 2. Replace $[\![x_A]\!]_N$, $[\![\boldsymbol{Q}]\!]_N$, $[\![\boldsymbol{P}]\!]_N$, $[\![\boldsymbol{P}]\!]_N$, $\{[\![c_j]\!]_N\}_j$ in Simulator 1 by uniformly random elements of the same type and compute $[\![v_j]\!]_{i^*}$ as $[\![v_j]\!]_{i^*} := -\sum_{i\neq i^*}[\![v_j]\!]_i$ for all $j \in [t]$. We note that the obtained simulator is independent of the witness x_A and solely takes the challenges $\{r_j, \boldsymbol{\varepsilon}_j\}_{j\in[t]}$ and i^* as input.

If $i^* = N$, the change only impacts the shares $\{[\![\boldsymbol{\alpha}_j]\!]_{i^*}\}_j$, $\{[\![\boldsymbol{\beta}_j]\!]_{i^*}\}_j$ and $\{[\![v_j]\!]_{i^*}\}_j$ in the simulated responses. We observe that the distributions of those shares are identical in Simulator 2 as in Simulator 1. Indeed, in both cases, the shares $\{[\![\boldsymbol{\alpha}_j]\!]_{i^*}\}_j$ and $\{[\![\boldsymbol{\beta}_j]\!]_{i^*}\}_j$ are uniformly distributed and independent of the rest and the shares $\{[\![v_j]\!]_{i^*}\}_j$ are defined by $[\![v_j]\!]_{i^*} = -\sum_{i\neq i^*}[\![v_j]\!]_i$.

If $i^* \neq N$, the change only impacts $\mathsf{aux} = ([\![x_A]\!]_N, [\![\boldsymbol{Q}]\!]_N, [\![\boldsymbol{P}]\!]_N, \{[\![c_j]\!]_N\}_j)$ in the simulated response (and the values derived from aux in the view of party N). We observe that the shares in aux are computed by adding one value of randomness from each seed from party $i \neq i^*$, then adding one value of randomness from party i^* (which is uniformly random in Simulator 1). Therefore aux is already uniformly random in Simulator 1 which implies that the output distributions of Simulator 1 and Simulator 2 are identical.

Simulator 3 (internal HVZK simulator). The only difference between Simulator 2 and the internal HVZK simulator described above is that the latter directly draws $\{[\![\boldsymbol{\alpha}_j]\!]_{i^*}\}_j$ and $\{[\![\boldsymbol{\beta}_j]\!]_{i^*}\}_j$ uniformly at random. As explained above, this does not impact the output distribution.

To sum up, we have shown that the internal simulator $\mathcal{S}$ outputs responses $(\text{Rsp}_1, \text{Rsp}_2)$ which are $(t, \varepsilon_{\text{PRG}})$-instinguishable from the responses in a real transcript of Protocol 2 (with same challenges). To get a global HVZK simulator, we proceed as follows:

1. Sample
   - $\text{Ch}_1 = \{r_j, \boldsymbol{\varepsilon}_j\}_{j\in[t]} \leftarrow \mathbb{F}_{\text{points}} \times \mathbb{F}_{\text{points}}^d$,
   - $\text{Ch}_2 = i^* \leftarrow [N]$.

   uniformly at random (as an honest verifier).
2. Run the internal simulator $\mathcal{S}(\{r_j, \boldsymbol{\varepsilon}_j\}_{j\in[t]},\ i^*)$ to get
   - $\text{Rsp}_1 = \text{Hash}([\![\boldsymbol{\alpha}_1]\!], [\![\boldsymbol{\beta}_1]\!], [\![v_1]\!], \ldots, [\![\boldsymbol{\alpha}_t]\!], [\![\boldsymbol{\beta}_t]\!], [\![v_t]\!])$,
   - $\text{Rsp}_2 = \big((\mathsf{state}_i, \rho_i)_{i\neq i^*} \mid \mathsf{com}_{i^*} \mid \{[\![\boldsymbol{\alpha}_j]\!]_{i^*}\}_{j\in[t]} \mid \{[\![\boldsymbol{\beta}_j]\!]_{i^*}\}_{j\in[t]}\big)$.
3. Compute the initial commitment Com as follows
   - For each party $i \neq i^*$, compute the commitment $\mathsf{com}_i = \text{Com}(\mathsf{state}_i; \rho_i)$;
   - For the unopened party, sample a random commitment com_{i^*};
   - Set $\text{Com} = \text{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$.
4. Output the transcript $T = (\text{Com}, \text{Ch}_1, \text{Rsp}_1, \text{Ch}_2, \text{Rsp}_2)$.

When applying the hiding property of the commitment scheme on com_{i^*}, we get that the global HVZK simulator outputs a transcript which is $(t, \varepsilon_{\text{PRG}} + \varepsilon_{\text{Com}})$-instinguishable from a real transcript of Protocol 2. □

F Proof of Theorem 4 (Soundness of Protocol 2)

For the sake of simplicity, we assume that the commitment scheme is perfectly binding. (If the commitment scheme was computationally binding we would have to deal with additional cases where the extractor would produce a commitment collision.)

For any set of successful transcripts corresponding to the same commitment, with at least two different Round 4 challenges (i^*),

- either the revealed shares of $[\![x_A]\!]$, $[\![\boldsymbol{P}]\!]$ and $[\![\boldsymbol{Q}]\!]$ are not consistent, and then we find a hash collision (if the committed values are not the same, then the commitments cannot be the same since the commitment scheme is perfectly binding),
- or the openings are unique and hence the underlying witness $([\![x_A]\!], [\![\boldsymbol{P}]\!], [\![\boldsymbol{Q}]\!])$ is uniquely defined.

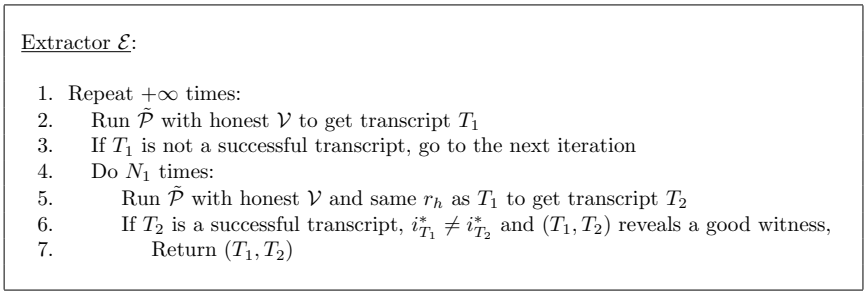
In the second case, this witness can be recovered from any two successful transcripts T_1 and T_2 corresponding to the same commitment and for which $i_1^* \neq i_2^*$. Let us call a witness $([\![x_A]\!], [\![\boldsymbol{P}]\!], [\![\boldsymbol{Q}]\!])$ a *good witness* whenever

$$\boldsymbol{S} \circ \boldsymbol{Q} = F \cdot \boldsymbol{P}$$

where $F := \prod_{j=0}^{m/d}(X - \gamma_j)$, $\boldsymbol{Q} := \sum_i [\![\boldsymbol{Q}]\!]_i$, $\boldsymbol{P} := \sum_i [\![\boldsymbol{P}]\!]_i$ and $\boldsymbol{S}$ is built by interpolation from x with $x := (x_A | y - H' x_A)$ and $x_A := \sum_i [\![x_A]\!]_i$. Such a witness enables us to build a solution for the syndrome decoding instance.

In what follows, we consider that the extractor only get transcripts with consistent shares since otherwise the extractor would find a hash collision.

We shall further denote by R_h the randomness of $\tilde{\mathcal{P}}$ which is used to generate the initial commitment $\textsc{Com} = h$, and we denote r_h a possible realization of R_h. Let us now describe the extractor procedure:



In what follows, we estimate the extraction complexity, *i.e.* how many time in average the extractor calls $\tilde{\mathcal{P}}$. Throughout the proof, we denote $\mathsf{succ}_{\tilde{\mathcal{P}}}$ the event that $\tilde{\mathcal{P}}$ succeeds in convincing $\mathcal{V}$. By hypothesis, we have $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}] = \tilde{\varepsilon}$.

Let us fix an arbitrary value $\alpha \in (0, 1)$ such that $(1 - \alpha)\tilde{\varepsilon} > \varepsilon$, it exists since $\tilde{\varepsilon} > \varepsilon$. Let r_h be a possible realization of R_h. We will say that r_h is *good* if it is such that

$$\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \geq (1 - \alpha) \cdot \tilde{\varepsilon} \ . \tag{5}$$

By the Splitting Lemma 5 (see Appendix D) we have

$$\Pr[R_h \text{ good} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}] \geq \alpha \ . \tag{6}$$

Let assume we sample a successful transcript T_1 as in the Step 2 of the extractor $\mathcal{E}$ and let r_h be the underlying realization of R_h. Assume r_h is good. By definition, we have

$$\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \geq (1 - \alpha) \cdot \tilde{\varepsilon} > \varepsilon > \frac{1}{N}$$

implying that there must exist a successful transcript T_2 with $i_2^* \neq i_1^*$. As explained above, this implies that there exists a unique and well-defined witness $(\llbracket x_A \rrbracket, \llbracket \boldsymbol{P} \rrbracket, \llbracket \boldsymbol{Q} \rrbracket)$ corresponding to these transcripts (and to all the transcripts with same r_h).

We will show that if this witness is a *bad* witness (*i.e.* is not a good witness) then we have $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \leq \varepsilon$ meaning that r_h is *not* good. By contraposition, we get that if r_h is good, then the witness $(\llbracket x_A \rrbracket, \llbracket \boldsymbol{P} \rrbracket, \llbracket \boldsymbol{Q} \rrbracket)$ is a good witness. So let us assume that the witness $(\llbracket x_A \rrbracket, \llbracket \boldsymbol{P} \rrbracket, \llbracket \boldsymbol{Q} \rrbracket)$ in T_1 is a bad witness. This means that

$$\boldsymbol{S} \circ \boldsymbol{Q} \neq F \cdot \boldsymbol{P}$$

where $\boldsymbol{Q} := \sum_i \llbracket \boldsymbol{Q} \rrbracket_i$, $\boldsymbol{P} := \sum_i \llbracket \boldsymbol{P} \rrbracket_i$ and $\boldsymbol{S}$ is built by interpolation from x with $x := (x_A | y - H' x_A)$ and $x_A := \sum_i \llbracket x_A \rrbracket_i$. So there exists j such that $\boldsymbol{S}_j \circ \boldsymbol{Q}_j \neq F \cdot \boldsymbol{P}_j$. Let us denote FP the event that a geniune execution of the MPC protocol outputs a false positive, *i.e.* outputs a zero vector v. Let us also denote ℓ the degree of the polynomial $D := \boldsymbol{S}_j \circ \boldsymbol{Q}_j - \boldsymbol{P}_j \cdot F$ and Z_i the event that i among the t evaluations of the polynomial D (by $r_1, \ldots, r_t$) are zeros. We have

$$\begin{aligned}
\Pr[\mathsf{FP}] &= \sum_{i=0}^{t} \Pr[\mathsf{Z}_i] \cdot \Pr[\mathsf{FP} \mid \mathsf{Z}_i] \\
&= \sum_{i=0}^{t} \frac{\binom{\ell}{i} \cdot \binom{\Delta - \ell}{t-i}}{\binom{\Delta}{t}} \cdot \left(\frac{1}{\Delta}\right)^{t-i}
\end{aligned}$$

The probability for Z_i is given by the Schwart-Zippel Lemma (see Appendix C) and the probability for "$\mathsf{FP} \mid \mathsf{Z}_i$" corresponds to the probability that the product checking (see Section 2.7) fails $t - i$ times, for the $t - i$ evaluation points $\{r_j\}$ for which $D(r_j) \neq 0$. Since the degree of D is less than $m' + w'$ (where $m' := \frac{m}{d}$ and $w' := \frac{w}{d}$), we get

$$\Pr[\mathsf{FP}] \leq \underbrace{\sum_{i=0}^{t} \frac{\max_{\ell \leq m'+w'-1} \left\{ \binom{\ell}{i} \cdot \binom{\Delta - \ell}{t-i} \right\}}{\binom{\Delta}{t}} \cdot \left(\frac{1}{\Delta}\right)^{t-i}}_{=:p}.$$

Let us upper bound the probability that the inner loop finds a successful transcript:

$$\begin{aligned}
\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] &= \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}, \mathsf{FP} \mid R_h = r_h] + \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}, \neg\mathsf{FP} \mid R_h = r_h] \\
&\leq p + (1 - p) \cdot \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h, \neg\mathsf{FP}]
\end{aligned}$$

Having a successful transcript means that the sharing $[\![v]\!]$ in the first response of the prover must encode a zero vector. But the event $\neg\mathsf{FP}$ when we have a bad witness implies that a geniune execution outputs a *non-zero* vector v. So to have a successful transcript, the prover must cheat for the simulation of at least one party. If the prover cheats for several parties, there is no way it can produce a successful transcript, while if the prover cheats for exactly one party (among the N parties), the probability to be successful is at most $1/N$. Thus, $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h, \neg\mathsf{FP}] \leq 1/N$ and we have

$$\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \leq p + (1 - p) \cdot \frac{1}{N} = \varepsilon,$$

meaning that r_h is *not* good. By contraposition, we get that if r_h is good, then $(\llbracket x_A \rrbracket, \llbracket \boldsymbol{P} \rrbracket, \llbracket \boldsymbol{Q} \rrbracket)$ is a good witness.

Now, let us lower bound the probability that the ith iteration of the inner loop finds a successful transcript T_2 such that $i^*_{T_1} \neq i^*_{T_2}$ in the presence of a good R_h. We have

$$\begin{aligned}
&\Pr[\mathsf{succ}^{T_2}_{\tilde{\mathcal{P}}} \cap (i^*_{T_1} \neq i^*_{T_2}) \mid R_h \text{ good}] \\
&\qquad = \Pr[\mathsf{succ}^{T_2}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}] - \Pr[\mathsf{succ}^{T_2}_{\tilde{\mathcal{P}}} \cap (i^*_{T_1} = i^*_{T_2}) \mid R_h \text{ good}] \\
&\qquad \geq (1-\alpha)\tilde{\varepsilon} - \Pr[i^*_{T_1} = i^*_{T_2} \mid R_h \text{ good}] \\
&\qquad = (1-\alpha)\tilde{\varepsilon} - \Pr[i^*_{T_1} = i^*_{T_2}] \\
&\qquad = (1-\alpha)\tilde{\varepsilon} - 1/N \\
&\qquad \geq (1-\alpha)\tilde{\varepsilon} - \varepsilon
\end{aligned}$$

Let define $p_0 := (1-\alpha)\cdot\tilde{\varepsilon} - \varepsilon$. By running $\tilde{\mathcal{P}}$ with the same r_h as for the good transcript N_1 times, we hence obtain a second non-colliding transcript T_2 with probability at least $1/2$ when

$$N_1 \approx \frac{\ln(2)}{\ln\left(\frac{1}{1-p_0}\right)} \leq \frac{\ln(2)}{p_0}\ . \tag{7}$$

Let C denotes the number of calls to $\tilde{\mathcal{P}}$ made by the extractor before finishing. While entering a new iteration:

- the extractor makes one call to $\tilde{\mathcal{P}}$ to obtain T_1,
- if T_1 is not successful, which occurs with probability $(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}])$,
  - the extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$,
- if T_1 is successful, which occurs with probability $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}]$,
  - if r_h is good which occurs with probability α, the extractor makes at most N_1 calls to $\tilde{\mathcal{P}}$ in the inner loop of $\mathcal{E}$ and output a pair (T_1, T_2) with probability $1/2$,
  - otherwise the extractor makes N_1 calls to $\tilde{\mathcal{P}}$ in the inner loop of $\mathcal{E}$ without stopping, with probability at most $(1 - \frac{\alpha}{2})$.

The mean number of calls to $\tilde{\mathcal{P}}$ hence satisfies the following inequality:

$$\mathbb{E}[C] \leq 1 + \underbrace{(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}]) \cdot \mathbb{E}[C]}_{T_1 \text{ unsuccessful}} + \underbrace{\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}] \cdot \left(N_1 + \left(1 - \frac{\alpha}{2}\right) \cdot \mathbb{E}[C]\right)}_{T_1 \text{ successful}}$$

which gives

$$\begin{aligned}
\mathbb{E}[C] &\leq 1 + (1-\tilde{\varepsilon}) \cdot \mathbb{E}[C] + \tilde{\varepsilon} \cdot \left(N_1 + \left(1 - \frac{\alpha}{2}\right) \cdot \mathbb{E}[C]\right) \\
&\leq 1 + \tilde{\varepsilon} \cdot N_1 + \mathbb{E}[C] \cdot \left(1 - \frac{\tilde{\varepsilon} \cdot \alpha}{2}\right) \\
&\leq \frac{2}{\alpha \cdot \tilde{\varepsilon}} \cdot (1 + \tilde{\varepsilon} \cdot N_1) \\
&\leq \frac{2}{\alpha \cdot \tilde{\varepsilon}} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{\ln(2)}{(1-\alpha)\cdot\tilde{\varepsilon} - \varepsilon}\right)
\end{aligned}$$

To obtain an α-free formula, let us take α such that $(1-\alpha)\cdot\tilde{\varepsilon} = \frac{1}{2}(\tilde{\varepsilon} + \varepsilon)$. We have $\alpha = \frac{1}{2}\left(1 - \frac{\varepsilon}{\tilde{\varepsilon}}\right)$ and the average number of calls to $\tilde{\mathcal{P}}$ is upper bounded by

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{2 \cdot \ln(2)}{\tilde{\varepsilon} - \varepsilon}\right)$$

which concludes the proof.

G Security Proof of the Signature Scheme

We give hereafter the proof of Theorem 5. This proof is highly inspired (and carbon copied where relevant) from the proof of the Picnic signature scheme [CDG+, Theorem 6.2].

Proof (Theorem 5). Let us consider a chosen-message EUF adversary $\mathcal{A}$ against the signature scheme. Let q_s denote the number of signing queries made by $\mathcal{A}$; let q_0, q_1, q_2, respectively, denote the number of queries to Hash_0, Hash_1 and Hash_2 made by $\mathcal{A}$. To prove security we define a sequence of experiments involving $\mathcal{A}$, where the first corresponds to the experiment in which $\mathcal{A}$ interacts with the real signature scheme. We let $\Pr_i[\cdot]$ refer to the probability of an event in experiment i. We let t denote the running time of the entire experiment, *i.e.*, including both $\mathcal{A}$'s running time and the time required to answer signing queries and to verify $\mathcal{A}$'s output.

Experiment 1. This corresponds to the interaction of $\mathcal{A}$ with the real signature scheme. In more detail: first KeyGen is run to obtain (H, y, x), and $\mathcal{A}$ is given the public key (H, y). In addition, we assume the random oracles Hash_0, Hash_1 and Hash_2 are chosen uniformly from the appropriate spaces. $\mathcal{A}$ may make signing queries, which will be answered as in the signature algorithm; $\mathcal{A}$ may also query any of the random oracles. Finally, $\mathcal{A}$ outputs a message/signature pair; we let Forge denote the event that the message was not previously queried by $\mathcal{A}$ to its signing oracle, and the signature is valid. Our goal is to upper-bound $\Pr_1[\mathsf{Forge}]$.

Experiment 2. We abort the experiment if, during the course of the experiment, a collision in Hash_0 is found. The number of queries to any oracle throughout the experiment (by either the adversary or the signing algorithm) is at most $(q_0 + \tau N q_s)$. Thus,

$$|\Pr_1[\mathsf{Forge}] - \Pr_2[\mathsf{Forge}]| \leq \frac{(q_0 + \tau N q_s)^2}{2 \cdot 2^{2\lambda}}.$$

Experiment 3. We abort the experiment if, during the course of the experiment, while answering to a signature query, the sampled salt collides with the value salt in any previous query to Hash_0, Hash_1 or Hash_2. For each single signature query, the probability to abort is upper bounded by $(q_s + q_0 + q_1 + q_2)/2^{2\lambda}$. Thus,

$$|\Pr_2[\mathsf{Forge}] - \Pr_3[\mathsf{Forge}]| \leq \frac{q_s \cdot (q_s + q_0 + q_1 + q_2)}{2^{2\lambda}} \ .$$

Experiment 4. The difference with the previous experiment is that, when signing a message m we begin by choosing h_1 and h_2 uniformly and then we expand them as $\{r_j^{[e]}, \varepsilon_j^{[e]}\}_{e\in[\tau],j\in[t]}$ and $\{i^{*[e]}\}_{e\in[\tau]}$. Phases 1, 3 and 5 of the signing algorithm are computed as before, but in phases 2 and 4 we simply set the output of Hash_1 to h_1 and the output of Hash_2 to h_2.

The outcome of this experiment compared to the previous one only changes if, in the course of answering a signing query, the query to Hash_1 or the query to Hash_2 was ever made before (by either the adversary or as part of answering some other signing query). But this cannot happen since in such a case Experiment 3 would abort. Thus,

$$\Pr_3[\mathsf{Forge}] = \Pr_4[\mathsf{Forge}] \ .$$

Experiment 5. The difference with the previous experiment is that, for each $e \in [\tau]$, we sample $\mathsf{com}_{i^{*[e]}}^{[e]}$ uniformly at random (*i.e.*, without making the corresponding query to Hash_0).

The only difference between this experiment and the previous one occurs if, during the course of answering a signing query, $\mathsf{seed}_{i^{*[e]}}^{[e]}$ (for some $e \in [\tau]$) was previously queried to Hash_0. However, such collision cannot occur within the same signing query (since the indices i and e are part of the input of Hash_0) and if it

occurs from a previous query (signing query or Hash_0 query) then the experiment aborts (according to the difference introduced in Experiment 3). Therefore,

$$\Pr_5[\mathsf{Forge}] = \Pr_4[\mathsf{Forge}] \ .$$

Experiment 6. We again modify the experiment. Now, for $e \in [\tau]$ the signer uses the internal HVZK simulator (see the proof E of Theorem 3) to generate the views of the parties in an execution of Phases 1 and 3. We denote $\mathcal{S}_{\mathsf{salt}}(\cdot)$ a call to this simulator which append salt to the sampled seed in input to TreePRG. This simulation results in $\{\mathsf{state}_i\}_{i \neq i^{*[e]}}$ and $(\llbracket \boldsymbol{\alpha}_j^{[e]} \rrbracket, \llbracket \boldsymbol{\beta}_j^{[e]} \rrbracket, \llbracket v_j^{[e]} \rrbracket)_{j \in [t]}$. Thus, signature queries are now answered as depicted in Figure 3.

Phase 0.

1. Sample h_1 uniformly from $\{0,1\}^{2\lambda}$.
2. Extend hash $\{r_j^{[e]}, \boldsymbol{\epsilon}_j^{[e]}\}_{e \in [\tau], j \in [t]} \leftarrow \mathrm{PRG}(h_1)$ where $r_j^{[e]} \in \mathbb{F}_{\mathrm{points}}$ and $\boldsymbol{\epsilon_j}^{[e]} \in \mathbb{F}_{\mathrm{points}}^d$.
3. Sample h_2 uniformly from $\{0,1\}^{2\lambda}$.
4. Expand hash $\{i^{*[e]}\}_{e \in [\tau]} \leftarrow \mathrm{PRG}(h_2)$ where $i^{*[e]} \in [N]$.
5. Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.

Phases 1 and 3. For each iteration $e \in [\tau]$,

1. $\{\mathsf{state}_i\}_{i \neq i^{*[e]}}, (\llbracket \boldsymbol{\alpha}_j^{[e]} \rrbracket, \llbracket \boldsymbol{\beta}_j^{[e]} \rrbracket, \llbracket v_j^{[e]} \rrbracket)_{j \in [t]} \leftarrow \mathcal{S}_{\mathsf{salt}}(\{r_j^{[e]}, \boldsymbol{\epsilon}_j^{[e]}\}_{j \in [t]}, i^{*[e]})$
2. Choose uniform $\mathsf{com}_{i^{*[e]}}^{[e]} \in \{0,1\}^{2\lambda}$. For other i, set $\mathsf{com}_i^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{[e]})$.

Phases 2 and 4.

1. Set $\mathrm{Hash}_1(m, \mathsf{salt}, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_{N-1}^{[\tau]}, \mathsf{com}_N^{[\tau]})$ equal to h_1.
2. Set $\mathrm{Hash}_2(m, \mathsf{salt}, h_1, \{\llbracket \boldsymbol{\alpha}_j^{[e]} \rrbracket, \llbracket \boldsymbol{\beta}_j^{[e]} \rrbracket, \llbracket v_j^{[e]} \rrbracket\}_{j \in [t], e \in [\tau]})$ equal to h_2.

Phase 5: Building of the signature. Output the signature σ built as

$$\mathsf{salt} \mid h_1 \mid h_2 \mid \Big((\mathsf{state}_i^{[e]})_{i \neq i^{*[e]}} \mid \mathsf{com}_{i^{*[e]}}^{[e]} \mid \{\llbracket \boldsymbol{\alpha}_j^{[e]} \rrbracket_{i^{*[e]}}\}_{j \in [t]} \mid \{\llbracket \boldsymbol{\beta}_j^{[e]} \rrbracket_{i^{*[e]}}\}_{j \in [t]} \Big)_{e \in [\tau]} \ .$$

Fig. 3: Experiment 6: Answer to a signature query for a message m.

Observe that the secret x is no longer used for generating signatures. Recall that an adversary against the internal HVZK simulator has distinguishing advantage $\varepsilon_{\mathrm{PRG}}$ (corresponding to execution time t), since commitments are built outside of the simulator. It results that

$$|\Pr_6[\mathsf{Forge}] - \Pr_5[\mathsf{Forge}]| \leq \tau \cdot q_s \cdot \varepsilon_{\mathrm{PRG}} \ .$$

Experiment 7. At any point during the experiment, we say that the execution e^* of a query

$$h_2 = \mathrm{Hash}_2(m, \mathsf{salt}, h_1, \{\llbracket \boldsymbol{\alpha}_j^{[e]} \rrbracket, \llbracket \boldsymbol{\beta}_j^{[e]} \rrbracket, \llbracket v_j^{[e]} \rrbracket\}_{j \in [t], e \in [\tau]})$$

defines *a correct witness* if the three following conditions are fulfilled:

(1) h_1 was output by a previous query

$$h_1 = \mathrm{Hash}_1(m, \mathsf{salt}, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_N^{[\tau]}) \ ,$$

(2) each $\mathsf{com}_i^{[e^*]}$ in this Hash_1-query was output by a previous query

$$\mathsf{com}_i^{[e^*]} = \mathrm{Hash}_0(\mathsf{salt}, e^*, i, \mathsf{state}_i^{[e^*]})$$

for every $i \in [N]$, and

(3) the vector x derived from $\{\mathsf{state}_i^{[e^*]}\}_{i\in[N]}$, *i.e.*

$$x = (x_A \mid x_B) \quad \text{with} \quad x_A := \sum\nolimits_i [\![x_A]\!]_i \quad \text{and} \quad x_B := y - H' x_A \ ,$$

satisfies $\mathrm{wt}(x) \leq w$.

(Note that in all cases the commitments in the relevant prior Hash_1-query, if it exists, must be unique since the experiment is aborted if there is ever a collision in Hash_0.)

In Experiment 7, for each query of the above form made by the adversary to Hash_2 (where m was not previously queried to the signing oracle), check if there exists an execution e^* which defines a correct witness. We let Solve be the event that this occurs for some query to Hash_2. Note that if that event occurs, the $\{\mathsf{state}_i^{[e^*]}\}$ (which can be determined from the oracle queries of the adversary) allow the computation of some vector x for which $Hx = y$ and $\mathrm{wt}(x) \leq w$. Thus, $\Pr_7[\mathsf{Solve}] \leq \varepsilon_{\mathrm{SD}}$.

We claim that

$$\Pr_7[\mathsf{Forge} \wedge \overline{\mathsf{Solve}}] \leq q_2 \cdot \varepsilon^\tau .$$

where $\varepsilon := p + \frac{1}{N} - p \cdot \frac{1}{N}$ is the soundness error of one execution. To see this, assume Solve does not occur. Then there is no execution of any Hash_2 query which defines a correct witness. When considering an arbitrary execution $e \in [\tau]$, the attacker can only possibly generate a forgery (using this Hash_2-query) if

1. $\boldsymbol{S}(r_j) \circ \boldsymbol{Q}(r_j) = \boldsymbol{P}(r_j) \cdot F(r_j)$ for all r_j (this occurs with probability p as defined in Equation (3)),
2. or there exists an r_j such that $\boldsymbol{S}(r_j) \circ \boldsymbol{Q}(r_j) \neq \boldsymbol{P}(r_j) \cdot F(r_j)$ and the value $i^{*[e]}$ is chosen to be the unique party such that the views of the remaining parties are consistent.

Thus, the overall probability with which the attacker can generate a forgery using this Hash_2-query is

$$\left(p + (1-p)\frac{1}{N}\right)^\tau \ .$$

The final bound is obtained by taking a union bound over all queries to Hash_2.

□