Shared Permutation for Syndrome Decoding: New Zero-Knowledge Protocol and Code-Based Signature*

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 are an important tool for many cryptographic protocols and applications. The threat of a coming quantum computer motivates the research for new zero-knowledge proof techniques for (or based on) post-quantum cryptographic problems. One of the few directions is code-based cryptography for which the strongest problem is the *syndrome decoding* (SD) of random linear codes. This problem is known to be NP-hard and the cryptanalysis state of affairs has been stable for many years. A zero-knowledge protocol for this problem was pioneered by Stern in 1993. As a simple public-coin three-round protocol, it can be converted to a post-quantum signature scheme through the famous Fiat-Shamir transform. The main drawback of this protocol is its high *soundness error* of 2/3, meaning that it should be repeated $\approx 1.7\lambda$ times to reach a λ-bit security.
In this paper, we improve this three-decade-old state of affairs by introducing a new zero-knowledge proof for the syndrome decoding problem on random linear codes. Our protocol achieves a soundness error of $1/n$ for an *arbitrary* n in complexity $\mathcal{O}(n)$. Our construction requires the verifier to *trust* some of the variables sent by the prover which can be ensured through a *cut-and-choose* approach. We provide an optimized version of our zero-knowledge protocol which achieves arbitrary soundness through parallel repetitions and merged cut-and-choose phase. While turning this protocol into a signature scheme, we achieve a signature size of 17 KB for a 128-bit security. This represents a significant improvement over previous constructions based on the syndrome decoding problem for random linear codes.

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: secure identification and signature, (anonymous) credentials, electronic voting, blockchain protocols and, more generally, privacy-preserving cryptography.

While many zero-knowledge proofs have been proposed for, or are based on, pre-quantum cryptographic problems (and in particular the discrete logarithm and the integer factoring), these protocols face the threat of obsolescence with the potential arising of a quantum computer in the coming decades [Sho94]. This motivates the question of finding efficient post-quantum alternatives to cryptographic protocols, and in particular, zero-knowledge proofs.

One of the few directions for these post-quantum alternatives is (error correcting) code-based cryptography. The *generic decoding* problem, or equivalently the *(computational) syndrome decoding* (SD) problem is a classic problem: given a matrix $H \in \mathbb{F}_2^{(m-k)\times m}$ and a vector $y \in \mathbb{F}_2^{m-k}$, recover a *small-weight* vector $x \in \mathbb{F}_2^m$ 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 parameters.

In a pioneering work from three decades ago [Ste94], Stern proposed a zero-knowledge protocol to prove the knowledge of a solution to a syndrome decoding instance. This protocol achieves a *soundness error* of 2/3

* The Version of Record of this article is published in *Designs, Codes and Cryptography*, and is available online at https://doi.org/10.1007/s10623-022-01116-1.

which means that a malicious prover can fool the verifier with probability $2/3$. Although an arbitrary security of $(2/3)^\tau$ can be achieved by repeating the protocol τ times, the induced communication cost is significant, which is partly due to this high soundness error. Since the work of Stern, a few papers have proposed optimizations and implementations (see for instance [Vér96,GG07,AGS11,ACBH13]) but for random linear codes with standard security levels, the communication cost is still heavy.

In the present paper, we propose a new zero-knowledge protocol for the SD problem which achieves a soundness error of $1/n$ with complexity $\mathcal{O}(n)$ for an *arbitrary* chosen n. In a nutshell, and as in Stern protocol, the solution x is masked by the application of a random permutation σ. However instead of revealing either $\sigma(x)$ or σ, we always reveal $\sigma(x)$ and prove the existence of a permutation σ. To this purpose, we decompose σ into n masked permutations $\sigma(\cdot) + s := (\sigma_1(\cdot) + s_1) \circ \cdots \circ (\sigma_n(\cdot) + s_n)$ which are all committed by the prover and we let the verifier choose $n-1$ of them to be revealed. This way, we can maintain the privacy of σ while obtaining the desired soundness error of $1/n$.

Our construction requires the verifier to *trust* some of the variables sent by the prover. This can be ensured by a so-called *cut-and-choose* phase since these variables are independent of the secret solution x. While composing our technique with a cut-and-choose phase, the obtained protocol has a similar structure as the zero-knowledge proof for Boolean circuits proposed by Katz, Kolesnikov and Wang [KKW18] as an efficient instantiation of the *MPC-in-the-head* paradigm [IKOS07]. We can therefore apply the same optimizations (such as the merging of the cut-and-choose phase) to obtain a 5-round zero-knowledge protocol with arbitrary soundness error $2^{-\lambda}$. If this protocol is used as an identification scheme with a impersonation probability lower than 2^{-16} (and SD security of 128 bits), the achieved communication cost is below 2 KB. Used as a zero-knowledge protocol with a soundness error of 2^{-128}, the communication cost can be made lower than 15 KB. We further detail how to make our zero-knowledge proof non-interactive and turn it into a signature scheme by applying the Fiat-Shamir transform [FS87,AABN02]. We provide a security proof of the obtained signature scheme in the random oracle model. For a 128-bit security level, our scheme achieves a signature size ranging between 17 KB (compact version) and 24 KB (fast version).

In the state of the art, two different directions are followed to build code-based signatures: applying the Fiat-Shamir transform to an identification scheme or using hash-and-sign paradigm with a code-based trapdoor function. In the former case, standard techniques results in identification schemes with high communication cost (implying large signatures) because of the non-negligible soundness error. The Schnorr-Lyubashevsky approach [Sch90,Lyu09] is a promising way to mitigate this cost, however in the case of code-based signatures it does not seem to work as well as for lattice-based signatures. In the hash-and-sign paradigm, existing schemes based on trapdoor functions are more sensitive to structural attacks. Such a signature scheme named Wave [DST19] has been proposed in 2018 and is still secured against the current cryptanalysis state of the art, but it suffers a huge public key and a slow signing time. Our signature has the advantage of being based on one of the oldest and hardest problem in code-based cryptography: syndrome decoding of random linear codes while still competing with existing schemes based on other (presumably weaker) problems in terms of public key and signature size.

The paper is organized as follows: In Section 2, we introduce the necessary background on the SD problem and zero-knowledge proofs. We present the basic protocol (achieving $\frac{1}{n}$ soundness) in Section 3 and an optimized version (achieving arbitrary soundness) in Section 4. Then, we describe a signature scheme obtained through the Fiat-Shamir transform in Section 5. To conclude, we provide performance estimations for different sets of parameters in Section 6 and compare our construction with other zero-knowledge proofs and signature schemes from the state of the art in Section 7.

2 Preliminaries

Throughout the paper, $\mathbb{F}_2$ shall denote the finite field with two elements. For any vector $x \in \mathbb{F}_2^m$, the *Hamming weight* of x, denoted $\mathrm{wt}(x)$, is the number of non-zero coordinates of x. For any $m \in \mathbb{N}^*$, the integer set $\{1, \ldots, m\}$ is denoted $[m]$. The set containing all the permutations of $[m]$ is denoted $\mathcal{S}_m$. For any matrix $H \in \mathbb{F}_2^{m \times n}$, the kernel of H, denoted $\mathrm{Ker}(H)$, is the set of solutions to the equation $Hx = 0$. 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 abreviated 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* if, for any algorithm $\mathcal{A}$ running in time at most t, we have

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

The two distributions are said

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

2.1 Syndrome Decoding Problem

Definition 1 (Syndrome Decoding Problem). *Let m, k and w be positive integers such that $m > k$ and $m > w$. The syndrome decoding problem with parameters (m, k, w) is the following problem:*

Let H, x and y be such that:

1. *H is uniformly sampled from $\mathbb{F}_2^{(m-k)\times m}$,*
2. *x is uniformly sampled from $\{x \in \mathbb{F}_2^m : \mathrm{wt}(x) = w\}$,*
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 (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:

$$w < \tau_{\mathrm{GV}}(m, k) \quad \Leftrightarrow \quad \binom{m}{w} < 2^{m-k} \ ,$$

we know that there exists a unique solution x such that $y = Hx$ with overwhelming probability. Otherwise, an instance has $\frac{1}{2^{m-k}} \cdot \binom{m}{w}$ 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.

2.2 Cryptographic Building Blocks

Definition 2 (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 $|E| \cdot \log(N/|E|)$ labels of the tree (which is presumably 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 $(\mathsf{seed}_i)_{i\notin E}$. In this paper, we shall denote $\mathsf{nodes}(\mathsf{mseed}, [N]\backslash E)$ the set of labels (intermediate seeds) necessary and sufficient to retreive the subseeds $(\mathsf{seed}_i)_{i\notin E}$ while still hiding $(\mathsf{seed}_i)_{i\in E}$.

Definition 3 (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*

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

A collision resistant hash function can be used to build a *Merkle tree* (a.k.a. *hash tree*). This is a binary tree in which every leaf node is labelled with the cryptographic hash of a data block v_i, and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. Given a collision-resistant hash function $\mathrm{Hash}(\cdot)$, the Merkle hash root for $N = 2^n$ input data blocks $v_1, \ldots, v_N$, denoted $\mathrm{Merkle}(v_1, \ldots, v_N)$, is hence defined as

$$\mathrm{Merkle}(v_1, \ldots, v_N) = \begin{cases} \mathrm{Hash}\left(\mathrm{Merkle}(v_1, \ldots, v_{N/2}) \parallel \mathrm{Merkle}(v_{N/2+1}, \ldots, v_N)\right) & \text{if } N > 1 \\ \mathrm{Hash}(v_1) & \text{if } N = 1 \end{cases}$$

A Merkle tree makes it possible to show the consistence of a small subset $E \subset [N]$ of revealed inputs $(v_i)_{i\in E}$ with the hash root $h = \mathrm{Merkle}(v_1, \ldots, v_N)$ without having to communicate all the other inputs $(v_i)_{i\notin E}$ (or their corresponding hash). The principle is to reveal the sibling paths of $(v_i)_{i\in E}$ in the Merkle tree, that we shall denote $\mathsf{auth}((v_1, \ldots, v_N), E)$, and which contains at most $|E| \cdot \log(N/|E|)$ hash values.

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

Definition 4 (Commitment Scheme). *A commitment scheme is a triplet of algorithms* $(\mathrm{KeyGen}, \mathrm{Com}, \mathrm{Verif})$ *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 5 (Correctness Property). *A commitment scheme achieves correctness, if for any message* x *and any randomness* ρ*:*

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

Definition 6 (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 7 (Binding Property). *A commitment scheme is binding if there exists a negligible function ν such that, for every (PPT) algorithm $\mathcal{A}$, we have*

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

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.*

2.3 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:

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

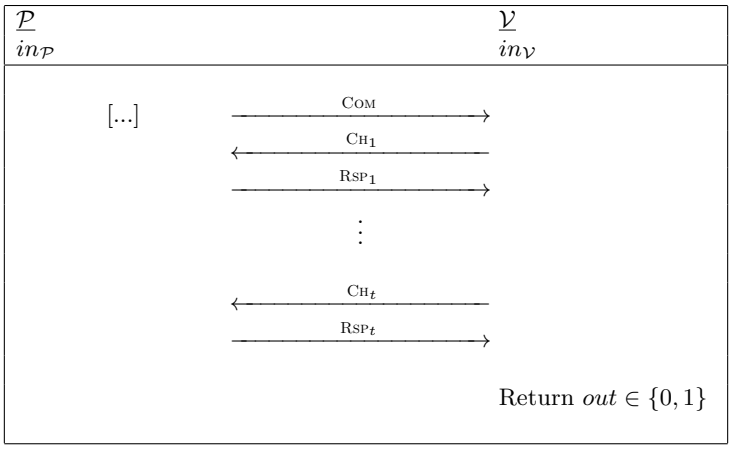
The sequence of exchanged messages is called the *transcript* of the execution, which is denoted

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

An execution producing an output *out* is further denoted

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

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 $\mathcal{A}$.



Protocol 1: Structure of a μ-round interactive proof with $\mu = 2t + 1$.

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_t, $\mathcal{V}$ produces a binary output: either 1, meaning that she was convinced by $\mathcal{P}$, or 0 otherwise. Such an μ-round interactive proof with $\mu = 2t + 1$ (1 commitment + t challenge-response pairs) is illustrated on Protocol 1.

In the protocols described in the following sections, the verifier shall make some calls to a checking procedure on input a Boolean expression of the form $(x = y)$, which is denoted Check $x = y$. This procedure stops the protocol execution and returns 0 (meaning that $\mathcal{V}$ is not convinced) if the Boolean expression evaluates to *false* (*i.e.* $x \neq y$ in the above example) while it does nothing if it evaluates to *true*: the protocol execution continues.

2.4 Zero-Knowledge Proofs

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 8 (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 vector x such that $y = Hx$ and $\mathrm{wt}(x) = w$.

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

Definition 9 (Zero-Knowledge Proof). *A proof of knowledge is {computa-tionally, statistically, perfectly} zero-knowledge if, for every malicious PPT verifier $\tilde{\mathcal{V}}$, there exists a PPT algorithm $\mathcal{S}$ (called simulator) which, given the input statement x and rewindable black-box access to $\tilde{\mathcal{V}}$, outputs a simulated transcript which is {computationally, statistically, perfectly} indistinguishable from the distribution* $\mathrm{View}(\langle \mathcal{P}(x, w), \tilde{\mathcal{V}}(x) \rangle)$.

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. By *honest-verifier* one means that the verifier messages are correctly sampled according to the protocol. In contrast, a malicious verifier could try to use another distribution for the messages in order to gain information about the witness. The (strong) zero-knowledge property requires that an execution of the protocol can be simulated without any knowledge of the witness for any (possibly malicious) verifier.

3 A Zero-Knowledge Protocol for Syndrome Decoding

Let us have an instance (H, y) of the syndrome decoding problem and let us denote x a solution of this instance.

3.1 General Idea

We now present the general idea of our protocol. As in the Stern protocol [Ste94], the prover first generates a random permutation σ and a random mask $r \in \mathrm{Ker}(H)$, and reveals

$$\big(v := \sigma(x)\ ,\ \tilde{x} := x + r\ \big) \tag{1}$$

to the verifier. The verifier can then verify that

- $y = H\tilde{x}$ holds: this ensures that $\tilde{x} = x + r$ for some mask $r \in \mathrm{Ker}(H)$,
- $\mathrm{wt}(v) = w$ holds: this ensures that $x = \sigma^{-1}(v)$ is of weight w for any permutation σ.

In order to complete the proof, the prover then needs to convince the verifier that there exists a pair (σ, r) which jointly satisfies:

1. $Hr = 0$, and

2. $\sigma(\tilde{x}) = v + \sigma(r)$.

The first property is independent of the solution x which makes it provable using a cut-and-choose approach (see details in Section 3.4). Our key idea is a way to prove the second property while achieving an arbitrary soundness error $1/n$.

Our method introduces an affine transformation $A(\cdot) = \sigma(\cdot) + s$ for a random mask s so that we have:

$$A(\tilde{x}) = \sigma(\tilde{x}) + s = \sigma(x) + \underbrace{\sigma(r) + s}_{q} \ . \tag{2}$$

If the verifier trusts that the value q equals $A(r)$ for some $r \in \mathrm{Ker}(H)$, then she only needs to verify the equality $A(\tilde{x}) = v + q$ to be convinced. Since q is independent of x, ensuring a trustworthy q can also be achieved through cut-and-choose (see details in Section 3.4). This leaves us with the task of proving $A(\tilde{x}) = v + q$.

Let note $u := A(\tilde{x})$. If the prover sends u, the verifier can check $u = q + v$. To prove $A(\tilde{x}) = u$, we decompose A into a composition of n affine transformations $A_i(\cdot) = \sigma_i(\cdot) + s_i$ such that $A = A_n \circ \cdots \circ A_1$. The permutation σ and mask s are then defined as

$$\sigma = \sigma_n \circ \cdots \circ \sigma_1 \quad \text{and} \quad s = s_n + \sigma_n(\ldots + \sigma_2(s_1)) \ . \tag{3}$$

The main utility of this decomposition is that revealing all the A_i except one gives no information on A, provided that A_i is uniformly sampled for every i. In this paradigm, the prover first commit all the A_i and reveals

$$\begin{aligned} u_1 &:= A_1(u_0) \\ u_2 &:= A_2(u_1) \\ &\cdots \\ u_n &:= A_n(u_{n-1}) \end{aligned}$$

where $u_0 := \tilde{x}$ and $u_n = u$ by definition. Then, the verifier chooses a random i^* such that the prover reveals all the A_i except A_{i^*}. The verifier can then check $u_i = A_i(u_{i-1})$ for every $i \in [n] \setminus \{i^*\}$. The only chance for the prover to cheat is to guess i^* in advance. Therefore, the maximum probability that a malicious prover can convince the verifier that $u = A(\tilde{x})$ is at most $1/n$.

To sum up,

1. The prover samples the random mask r from $\mathrm{Ker}(H)$ and n affine transformations $A_i(\cdot) := \sigma_i(\cdot) + s_i$. We denote $A(\cdot) := \sigma(\cdot) + s$ the composition $A_n \circ \cdots \circ A_1$, with σ and s satisfying (3).
2. The prover reveals $q := \sigma(r) + s$, $v := \sigma(x)$ and $\tilde{x} := x + r$.
3. The verifier checks $y = H\tilde{x}$ and $\mathrm{wt}(v) = w$.
4. The prover reveals $u_i := A_i(u_{i-1})$ for $i \in \{1, ..., n\}$, with $u_0 := \tilde{x}$;
5. The verifier checks $u_n = q + v$.
6. The prover commits all the A_i, *i.e* all the (σ_i, s_i);
7. The verifier generates $i^* \leftarrow [n]$ and sends it to the prover as challenge;
8. The prover reveals $\{A_i\}_{i \neq i^*}$.
9. The verifier checks $u_i = A_i(u_{i-1})$ for all $i \neq i^*$.

If all the checks have passed, the verifier deduces $u = A(\tilde{x})$. Moreover, if q is *trusted*, then a proof that $u = A(\tilde{x})$ constitutes a proof of knowledge of a solution x to the syndrome decoding instance. This protocol is a zero-knowledge protocol with a soundness error of $1/n$ under the trust hypothesis on q. As previously mentioned, this trust hypothesis can be fulfilled by using a cut-and-choose approach that we detail in Section 3.4.

3.2 Description of the Protocol

We give a formal description of our new zero-knowledge proof for syndrome decoding in Protocol 2. For the sake of simplicity, this description assumes that the value $q = \sigma(r) + s$ is *trusted* by the verifier. We denote this trust requirement with a star in superscript: $q^\star$.

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$x \in \mathbb{F}_2^m$ *s.t.* $\mathrm{wt}(x) = w$		
$H \in \mathbb{F}_2^{(m-k)\times m}$, $y := Hx$		H, $y := Hx$
For i in $\{1, \dots, n\}$:		
$\quad \rho_i \leftarrow \{0,1\}^\lambda$		
$\quad (\sigma_i, s_i) \leftarrow \mathcal{S}_m \times \mathbb{F}_2^m$		
$\quad c_i = \mathrm{Com}((\sigma_i, s_i); \rho_i)$		
$r \leftarrow \mathrm{Ker}(H)$		
$\sigma = \sigma_n \circ \cdots \circ \sigma_1$		
$s = s_n + \sigma_n(\dots + \sigma_2(s_1))$		
$q^\star = \sigma(r) + s$		
$v = \sigma(x)$		
$\tilde{x} = x + r$		
$u_0 = \tilde{x}$		
For i in $\{1, \dots, n\}$:		
$\quad u_i = \sigma_i(u_{i-1}) + s_i$		
	$\xrightarrow{c_1, \dots, c_n}$	
	$\xrightarrow{q^\star}$	
	$\xrightarrow{v, \tilde{x}}$	
	$\xrightarrow{u_1, \dots, u_n}$	$i^* \leftarrow \{1, \dots, n\}$
	$\xleftarrow{i^*}$	
	$\xrightarrow{(\sigma_i, s_i, \rho_i)_{i \neq i^*}}$	$u_0 = \tilde{x}$
		For all $i \neq i^*$:
		$\quad$ Check $\mathrm{Verif}((\sigma_i, s_i), c_i, \rho_i) = 1$
		$\quad$ Check $u_i = \sigma_i(u_{i-1}) + s_i$
		Check $u_n = v + q^\star$
		Check $\mathrm{wt}(v) = w$
		Check $y = H\tilde{x}$
		Return 1

Protocol 2: Zero-knowledge proof for syndrome decoding – Simplified version with trusted $q^\star$.

3.3 Security Proofs

The following theorems state the completeness, zero-knowledge and soundness (with trusted q) of Protocol 2. The proofs of Theorems 2 and 3 are provided in appendix.

Theorem 1 (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 2 (Zero-Knowledge). *Let the commitment scheme* Com *used in Protocol 2 be* {*computationally, statistically, perfectly*} *hiding. For all malicious PPT verifier $\tilde{\mathcal{V}}$, there exists a PPT simulator $\mathcal{S}$ which, given rewindable black-box access to $\tilde{\mathcal{V}}$, outputs a simulated transcript which is* {*computationally, statistically, perfectly*} *indistinguishable from a real transcript between $\mathcal{P}$ and $\tilde{\mathcal{V}}$.*

Theorem 3 (Soundness). *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input (H, y),*

- *builds q honestly, i.e. for any $\{(\sigma_i, s_i), \rho_i\}_i$ extractable from $\tilde{\mathcal{P}}$ and such that* $\mathrm{Verif}((\sigma_i, s_i), c_i, \rho_i) = 1$, *there exists $r \in \mathrm{Ker}(H)$ such that $q = \sigma(r) + s$ with $\sigma = \sigma_n \circ \cdots \circ \sigma_1$ and $s = s_n + \sigma_n(\dots + \sigma_2(s_1))$;*

– *convinces the honest verifier* $\mathcal{V}$ *on input* H, y *to accept with probability*

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

where the soundness error ε *is equal to* $1/n$.

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* $\mathrm{wt}(x) = w$*, or a commitment collision, by making in average*

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

calls to $\tilde{\mathcal{P}}$.

3.4 Producing the Trusted Vector

In order to obtain a sound zero-knowledge proof, we need a procedure to build the trusted vector q. One possible technique is to use the *cut-and-choose* methodology. Concretely, the prover generates many different vectors q in a *verifiable* way, *i.e.*, by committing the randomness used for their generation. Then, the verifier will ask the prover to reveal how she built some of the vectors q, namely to *open* these vectors q. This opening consists in revealing the corresponding $r \in \mathrm{Ker}(H)$ as well as the (previously committed) randomness used to generate the affine transformation $A(\cdot)$. The verifier can then check the consistency of the commitments, the belonging of r to $\mathrm{Ker}(H)$ and the correct computation of $q = A(r)$. Since the prover's secret (the solution x to the syndrome decoding instance) is not involved, such an opening does not break the zero-knowledge property of the protocol. However the opened vectors q become unusable for the protocol (because the used randomness is revealed). Thus, the prover must use one of the vectors q for which the building was not opened.

Let M be the number of generated and committed vectors. In the case of the simple protocol with soundness error $1/n$, one only needs a single trusted vector q. The verifier then asks for the opening of $M-1$ of the generated pairs before running the "trusted" part of the proof. If the prover cheated for more than one vector, then the verification will always fail. If the prover cheated on one vector, then the probability that the opening of this vector is not requested by the verifier during the cut-and-choose method is $1/M$.

The generation of a trusted vector q via the cut-and-choose method is described in Protocol 3.

This protocol can be composed with the sound phase (Protocol 2) to obtain a standalone 5-round protocol. This protocol is a zero-knowledge proof for the syndrome decoding problem with a soundness error of

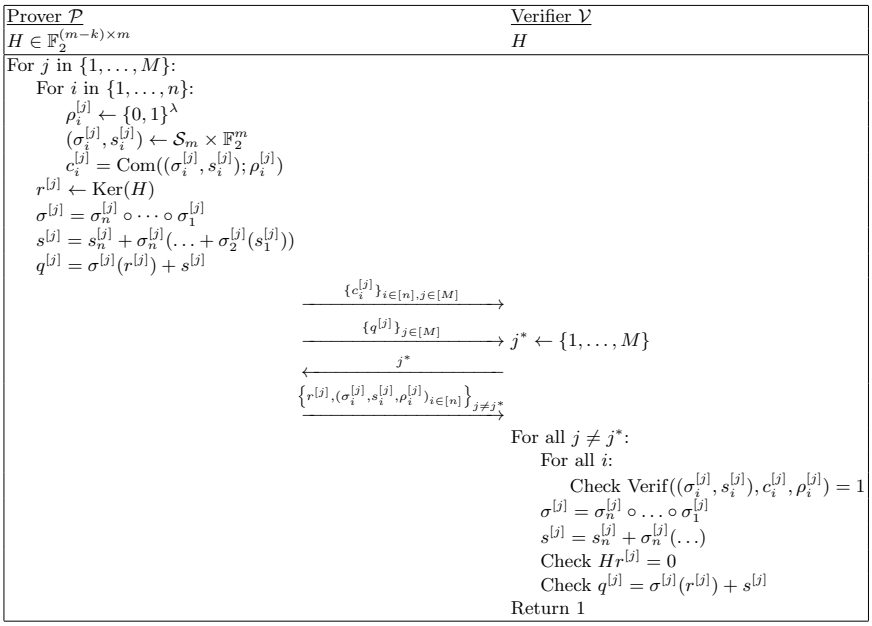
$$\max\left\{\frac{1}{M}, \frac{1}{n}\right\}.$$

Indeed, the prover has two possible ways to cheat: either she cheats for one of the M vectors during the cut-and-choose phase, but if this vector is not selected the protocol is aborted (the cheating is discovered); or she can try to cheat during the sound phase. The prover can try to introduce a mistake in the first phase ($\frac{1}{M}$) or in the second phase ($\frac{1}{n}$), but not in both.

For completeness, we depict this standalone version in Protocol 4.

Remark 2. Since the cut-and-choose does not require the prover secret, it can be executed before the prover gets its secret key. This step is hence a *preprocessing* phase of the sound protocol.

Remark 3. This work has been realized independently of the work of Beullens [Beu20], but our protocol enters in the definition of *protocol with helper* from [Beu20]. The latter definition formalizes the idea behind the preprocessing phase proposed in [KKW18].



Protocol 3: Cut-and-choose protocol to produce a trusted vector q.

4 The Five-Round Zero-Knowledge Protocol

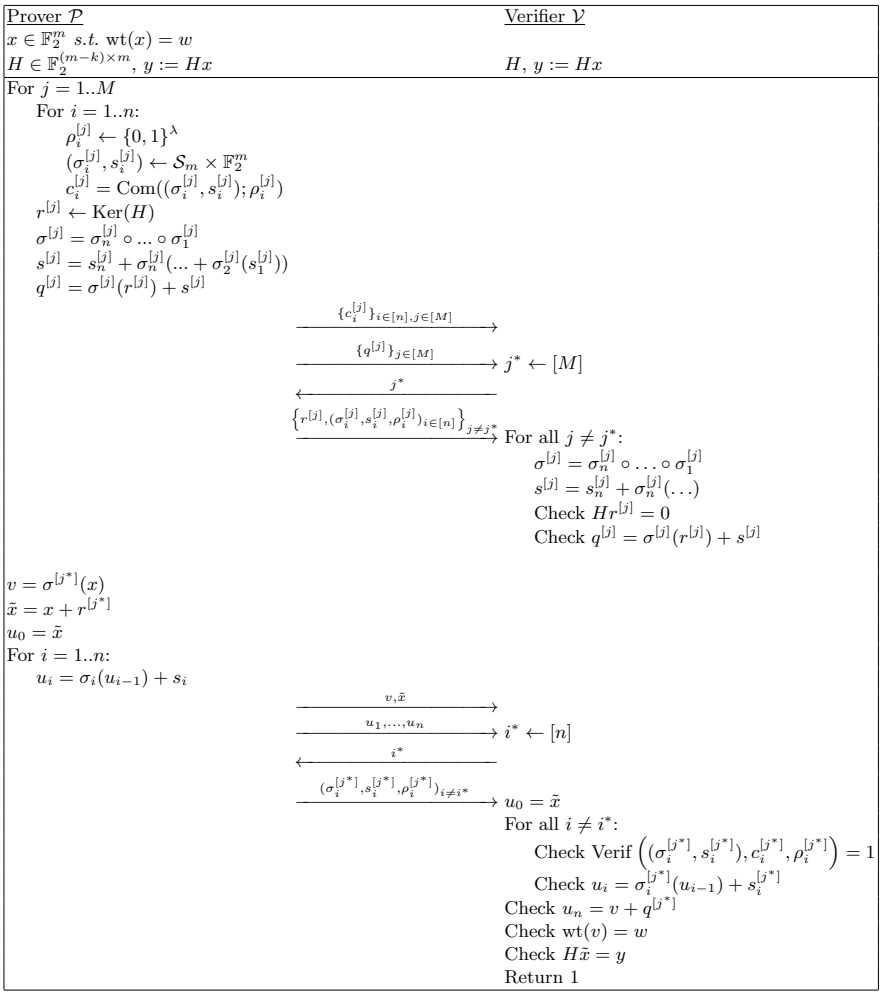
In the previous section, we described a new zero-knowledge protocol for the syndrome decoding problem with a soundness error of $\max\{1/n, 1/M\}$ and complexity $\mathcal{O}\big(n \cdot M \cdot \mathsf{poly}(\lambda)\big)$ for arbitrary chosen parameters n and M and with λ being the security level of the syndrome decoding instance. In order to obtain a soundness error close to $2^{-\lambda}$ for a target security parameter λ, one can run $\lambda/(\log_2 \min\{n, M\})$ independent repetitions of the protocol. However using such a simple approach is not optimal in terms of communication.

In this section, we present various optimizations to make our protocol efficient while targeting a standard security level. As a result, we describe a 5-round HVZK protocol that achieves $2^{-\lambda}$ soundness with optimized communication cost. For instance, for $\lambda = 128$, the size of the produced proof can be made lower than 15 KB.

4.1 Presentation of the Optimizations

We present here various optimizations to make our proof more compact. Our first improvement consists in merging the τ cut-and-choose phases (this was suggested in [KKW18] in a similar context). Then, we show how the pseudorandom generation of the precomputed values from compact seeds can save a lot of communication. This can be further improved by using pseudorandom generation trees for the seeds (as also suggested in [KKW18]). Finally, we show that some values can be recomputed by the verifier and can hence be trade for commitments in the prover messages. We further describe how to efficiently instantiate the commitment scheme using a collision-resistant hash function.

Merging of the cut-and-choose phase. As suggested in [KKW18], instead of repeating the cut-and-choose to obtain a trusted vector τ times, we can perform the cut-and-chose to generate τ trusted vectors all at once. Specifically, the prover $\mathcal{P}$ generate M vectors (for a large enough M) and the verifier requests the opening of $M - \tau$ of these vectors. After checking that the opened vectors have honestly been generated, the verifier trusts the remaining τ vectors which are then used for the τ independent executions of the sound phase.



Protocol 4: Standalone zero-knowledge proof for syndrome decoding

Let us assume that a malicious prover correctly builds k vectors and cheats for $M-k$ vectors. Without loss of generality, k satisfies $k \geq M - \tau$, otherwise the cheating prover will have to open a dishonest vector and the proof will fail. Then, the probability of this malicious prover to successfully passing the merged cut-and-choose phase is at most $\binom{k}{M-\tau} \cdot \binom{M}{M-\tau}^{-1}$. Let $\gamma = M - k \leq \tau$ denote the number of dishonest vectors. Conditioned on passing the first phase, her probability of passing the τ executions of the protocol is at most

$$\underbrace{1 \times ... \times 1}_{\substack{\gamma \text{ times} \\ \text{(upper bound of the} \\ \text{success probability} \\ \text{for the cheating pairs)}}} \quad \times \quad \underbrace{\frac{1}{n} \times ... \times \frac{1}{n}}_{\substack{\tau-\gamma \text{ times} \\ \text{(success probability for the} \\ \text{unchecked but honest pairs)}}} \quad = \frac{1}{n^{\tau-\gamma}} = \frac{1}{n^{k-M+\tau}} \ .$$

The soundness error is therefore

$$\varepsilon(M, n, \tau) := \max_{M-\tau \leq k \leq M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot n^{k-M+\tau}} \right\}.$$

Some examples of values for M, n and τ to reach a target security $\lambda \in \{128, 256\}$ are given in the Table 1.

n	4	8	16	32	64	128
M	256	293	512	606	842	1291
τ	64	43	32	26	22	19

(a) $\lambda = 128$

n	4	8	16	32	64	128
M	512	583	1024	1199	2144	3379
τ	128	86	64	52	43	37

(b) $\lambda = 256$

Table 1: Example values of M, n and τ to achieve statistical security $\lambda \in \{128, 256\}$. These parameters have been obtained while minimizing τ for a given n. Then given n and the underlying minimal τ, M is taken to be as small as possible. NB: Other strategies provide different trade-offs.

Working with seeds and PRG. In the basic protocol, the variables σ_i and s_i are uniformly sampled at random for every i. This imposes the prover to reveal $n-1$ pairs (σ_i, s_i) in its final response which is expensive since $s_i \in \mathbb{F}_2^m$ and $\sigma_i \in \mathcal{S}_m$. Instead, we can sample a random seed sseed_i and use a pseudo-random generator (PRG) to generate the pair (σ_i, s_i) from sseed_i. This way, whenever the prover wants to reveal (σ_i, s_i), she simply reveals seed_i. For a target security of λ bits, the seed only needs to make λ bits: we hence trade a message of $\log_2(m!) + m$ bits for one of λ bits. And to avoid revealing the commitment randomness ρ_i, the prover can derive it from another seed seed_i. But since the prover wants to reveal both ρ_i and sseed_i (to get the couple (σ_i, s_i)) when she opens the commitment, she can sample ρ_i and sseed_i from the same seed seed_i.

On the other hand, when a vector q is chosen by the verifier to be opened in the cut-and-choose phase, the prover must reveal (σ_i, s_i) for every $i \in [n]$ as well as r. In order to save further communication, all the seed_i, as well as the random mask r, can be generated from *master seed*: $\big(r, (\mathsf{seed}_i)_{i\in[n]}\big) \leftarrow \mathrm{PRG}(\mathsf{mseed})$. This way, when the vector q must be open, the prover sends mseed instead of $\big(r, (\mathsf{seed}_i)_{i\in[n]}\big)$ and replace $k + \lambda \cdot n$ bits of communication by λ bits.

Let $j \in \{1, ..., M\}$ denote the index of a precomputed set of variables during the cut-and-choose phase. In what follows, the master seed will be denoted $\mathsf{mseed}^{[j]}$, the n intermediary seeds will be denoted $\mathsf{seed}_i^{[j]}$ and the n subseeds will be denoted $\mathsf{sseed}_i^{[j]}$. Figure 1 illustrates the relation between the three types of seeds and the sampled variables.

For $j = 1..M$:
  $\big(r^{[j]}, (\mathsf{seed}_i^{[j]})_i\big) \leftarrow \mathrm{PRG}(\mathsf{mseed}^{[j]})$
  For $i = 1..n$:
    $\rho_i^j, \mathsf{sseed}_i^j \leftarrow \mathrm{PRG}(\mathsf{seed}_i^{[j]})$
    $(\sigma_i^{[j]}, s_i^{[j]}) \leftarrow \mathrm{PRG}(\mathsf{sseed}_i^{[j]})$

Fig. 1: Pseudorandom generation in the cut-and-choose phase.

Using generation trees. The previous optimization enables to reduce the communication by replacing large variables (vectors of $\mathbb{F}_2^m$ and $[m] \to [m]$ permutations) by compact seeds. As suggested in [KKW18], we can go one step further by reducing the number of seeds that must be revealed by the prover using a structured generation.

In the above description, we generate n intermediary seeds $\mathsf{seed}_i^{[j]}$ from a master seed $\mathsf{mseed}^{[j]}$. If all the intermediary seeds must be opened, then the prover only reveals the master seed, but if the index j is chosen for the second phase then only $n-1$ of them must be revealed, which implies $(n-1)\cdot\lambda$ bits of communication. In order to avoid this factor $(n-1)$ we can use a tree PRG (see description in Section 2.2). The principle is

to label the root of a binary tree of depth $\log_2 n$ with $\mathsf{mseed}^{[j]}$. Then, one inductively labels the children of each node with the output of a PRG applied to the node's label. The subseeds $(\mathsf{seed}_i^{[j]})_{i\in[n]}$ are defined as the labels of the n leaves of the tree. To reveal $(\mathsf{seed}_i^{[j]})_{i\neq i^*}$, it suffices to reveal the labels on the siblings of the path from the root of the tree to leaf i^*. Those labels allow the verifier to reconstruct $(\mathsf{seed}_i^{[j]})_{i\neq i^*}$ while still hiding $\mathsf{seed}_{i^*}^{[j]}$. Applying this optimization reduces the communication complexity to $\lambda \cdot \log_2 n$ for revealing the seeds $(\mathsf{seed}_i^{[j]})_{i\neq i^*}$. The same strategy can further be applied to the master seeds $(\mathsf{mseed}^{[j]})_{j\in[M]}$ in the cut-and-choose phase. By using a generation tree to generate the master seeds from a *grandmaster seed*, the communication cost of revealing $M-\tau$ master seeds out of M is decreased to $\lambda \cdot \tau \log_2 \frac{M}{\tau}$ bits (instead of $\lambda \cdot (M-\tau)$ bits).

For the sake of simplicity we shall omit this optimization from the description of our protocol. However, we stress that it can be applied as is without impact on our security statements.

Keeping the bare minimum. In the basic protocol, the prover sends all the vectors $\{q^{[j]}\}$. However those are large m-bit vectors which can actually be recomputed by the verifier. Indeed,

- if j is among the opened indexes in the cut-and-choose phase, the verifier has access to the master seed $\mathsf{mseed}^{[j]}$. Thus, she can re-sample the values $r^{[j]}$, $(\sigma_i^{[j]}, s_i^{[j]})_i$ from which $q^{[j]}$ can be recomputed;
- if j is not among the opened indexes, then the verifier has the following relation: $u_n^{[j]} = v^{[j]} + q^{[j]}$, so that $q^{[j]}$ can be re-computed as $q^{[j]} = u_n^{[j]} - v^{[j]}$.

The verifier still needs to check that the prover knew the values of $\left\{q^{[j]}\right\}$ at the beginning of the interaction. That is, sending $\left\{q^{[j]}\right\}$ is replaced by sending commitments of $\left\{q^{[j]}\right\}$.

Another similar optimization can be applied in the second phase of the protocol. In the basic version, the prover sends all the u_i to the verifier but since the latter eventually knows $(\sigma_i, s_i)_{i\neq i^*}$, she can recompute all the u_i's, except u_{i^*}, thanks to the relation: $u_i = \sigma(u_{i-1}) + s_i$. Therefore, the prover only needs to send u_{i^*} to the verifier. Once again, such a modification implies that the u_i must be committed by the prover before receiving i^*.

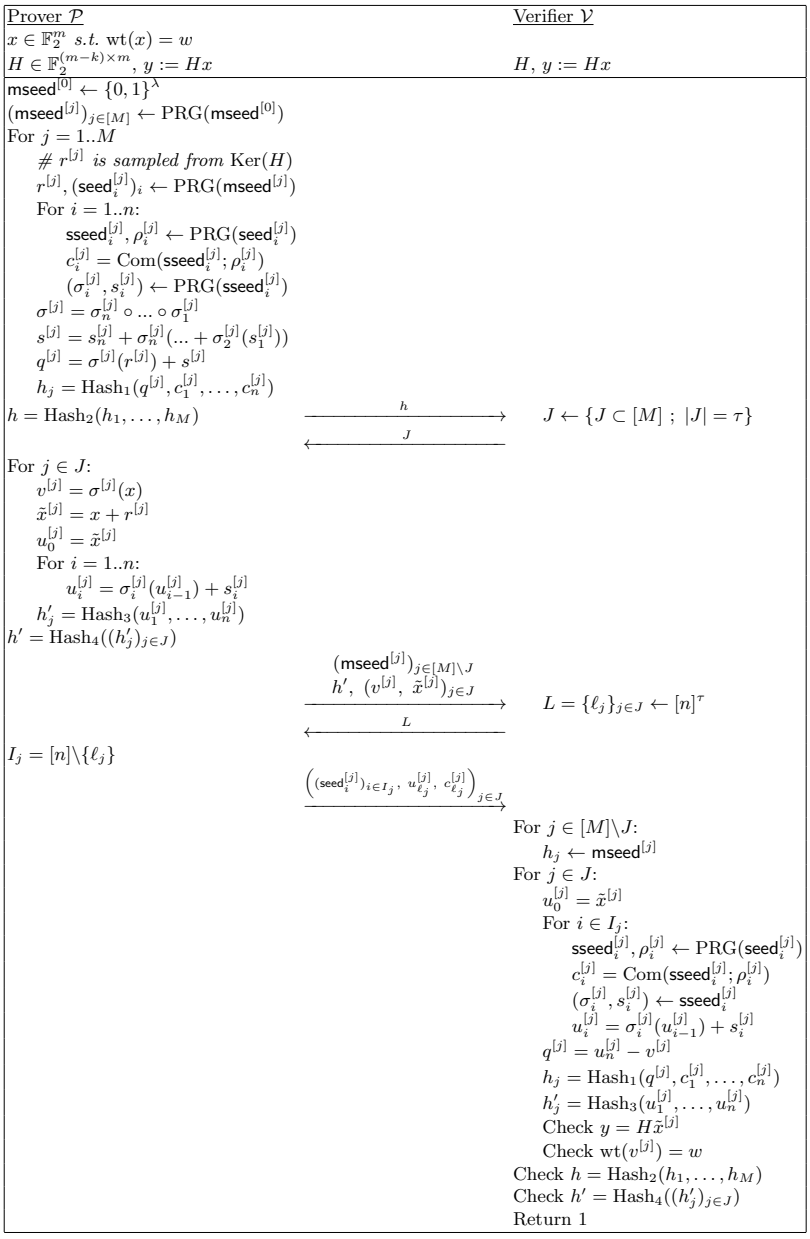
Hash-based commitments. Since we strive at simplicity and efficiency, whenever possible we use a simple hash-based commitment scheme defined by $\mathrm{Com} : x \mapsto \mathrm{Hash}(x)$ and $\mathrm{Verif} : (x, c) \mapsto (c = \mathrm{Hash}(x))$. Such a scheme is known to be computationally binding under the collision-resistance of Hash, but not computationally hiding.

To keep the zero-knowledge property for the protocol, the commitments $c_i^{[j]}$ on the pairs $(\sigma_i^{[j]}, s_i^{[j]})$ must be hiding. For those, we must hence keep a hiding commitment scheme.[4] However for the other commitments, the inputs are not secret: the vector $q^{[j]}$ is committed with the corresponding $(c_i^{[j]})_i$ in a common hash-based commitment $h_j := \mathrm{Hash}(q^{[j]}, c_1^{[j]}, \ldots, c_n^{[j]})$. All these h_j commitments are further regrouped into a single hash value $h := \mathrm{Hash}(h_1, \ldots, h_M)$ which forms the initial commitment of the prover. During the second phase, after receiving the set $J \subseteq [M]$ of the τ trusted indexes as challenge from the verifier, the prover commits all the values $(u_i^{[j]})_{i\in[n],j\in J}$ into a single commitment $h' := \mathrm{Hash}((u_i^{[j]})_{i\in[n],j\in J})$.

4.2 Description of the Protocol

After applying the various optimizations described above, we obtain a 5-round HVZK protocol with much more compact computation which is depicted in Protocol 5. The protocol makes use of one commitment scheme Com and four hash functions Hash_1, Hash_2, Hash_3 and Hash_4, whose output ranges are assumed to be consistent with the protocol description.

[4] We might for instance use a computationally hiding hash-based commitment scheme defined as $\mathrm{Com} : (x, \rho) \mapsto \mathrm{Hash}(x \parallel \rho)$ for a long-enough random nonce ρ.



Protocol 5: Five-round HVZK proof for the syndrome decoding problem.

4.3 Security Proofs

We prove hereafter that Protocol 5 achieves completeness, honest-verifier zero-knowledge, and $2^{-\lambda}$-soundness (for appropriately chosen parameter M, n and τ). The proofs of Theorems 5 and 6 are provided in appendix.

Theorem 4 (Completeness). *Protocol 5 is perfectly complete.*

Proof. For any sampling of the random coins of $\mathcal{P}$ and $\mathcal{V}$, if the computation described in Protocol 5 is genuinely performed then all the checks of $\mathcal{V}$ pass. □

Theorem 5 (Honest-Verifier Zero-Knowledge). *Let the PRG used in Protocol 5 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*

challenges J and L outputs a transcript which is $(t, \varepsilon_{PRG} + \varepsilon_{Com})$-indistinguishable from a real transcript of Protocol 5.

Theorem 6 (Soundness of Protocol 5). *Let*

$$\varepsilon := \max_{M-\tau \leq k \leq M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot n^{k-M+\tau}} \right\} . \tag{4}$$

Suppose there is an efficient prover $\tilde{\mathcal{P}}$ such that

$$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle(H, y) \to 1] > \varepsilon , \tag{5}$$

where (H, y) is a random syndrome decoding instance. Then, there exists an extraction algorithm $\mathcal{E}$ which, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces with either a witness x such that $y = Hx$ and $\mathrm{wt}(x) = w$, or a commitment collision, by making in average

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{8M}{\tilde{\varepsilon} - \varepsilon} \right)$$

calls to $\tilde{\mathcal{P}}$.

5 The Signature Scheme

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

5.1 Transformation into a Non-Interactive Scheme

With a straight application of Fiat-Shamir to Protocol 5, we would compute the challenges J and L as:

$$J := \mathrm{Hash}'_1(m, h)$$

and

$$L := \mathrm{Hash}'_2\left(m, h, (\mathsf{mseed}^{[j]})_{j \in [M] \setminus J}, (v^{[j]})_{j \in J}, (\tilde{x}^{[j]})_{j \in J}, h'\right) ,$$

where m is the input message and where Hash'_1 and Hash'_2 are some hash functions.

But doing so would imply an overhead on the size of the challenges J and L for the security to hold. 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

$$\mathrm{cost}_{\mathrm{forge}} := \min_{M-\tau \leq k \leq M} \left\{ \frac{\binom{M}{M-\tau}}{\binom{k}{M-\tau}} + n^{k-M+\tau} \right\} .$$

which is substantially lower than the target forgery cost ε^{-1}, for ε being the soundness error of Protocol 5 (see Theorem 6).

A numerical analysis of the parameters shows that we get a more efficient signature scheme by turning Protocol 5 into a 3-round protocol before applying the Fiat-Shamir transform. Instead of waiting the challenge J from the verifier, the prover can directly commit $v^{[j]}$, $\tilde{x}^{[j]}$ and $\{u_i^{[j]}\}$ for all $j \in \{1, \dots M\}$. Thus the verifier can send both challenges J and L at the same time. For this purpose, the prover will compute

$$h'_j = \mathrm{Hash}(v^{[j]}, \tilde{x}^{[j]}, u_1^{[j]}, \dots, u_n^{[j]})$$

for all $j \in \{1, \dots, M\}$ and she will send $h' = \mathrm{Hash}(h'_1, \dots, h'_M)$ to the verifier. After receiving challenge (J, L), the prover sends h'_j for $j \notin J$ to enable the verifier to rebuild h'. In order to decrease the cost of sending all these h'_j, she can use a Merkle tree: after committing $h' = \mathrm{Merkle}(h'_1, \dots, h'_M)$, proving the consistence of $(h'_j)_{j \in J}$ with the hash root h' can be done by revealing at most $\tau \cdot \log_2\left(\frac{M}{\tau}\right)$ labels of the Merkle tree (instead of $M - \tau$ labels).

The resulting 3-round protocol is also an honest-verifier zero-knowledge protocol with the same soundness. It can indeed be checked that the described modification has essentially no impact on the proofs of Theorem 5 (honest verifier zero-knowledge) and Theorem 6 (soundness). While the communication cost is slightly greater for this 3-round version than for the original 5-round protocol, the transformation into a non-interactive scheme does not suffer the aforementioned attack, which allows much better parameters. Now the application of Fiat-Shamir applies to compute the challenges J and L as

$$(J, L) := \mathrm{Hash}'(m, h, h') \ .$$

Moreover since we have $h = \mathrm{Hash}_2(h_1, \dots, h_M)$ and the h_j's are known by the verifier, we can directly compute the challenges J and L as:

$$(J, L) := \mathrm{Hash}'(m, h_1, \dots, h_M, h') \ .$$

On the other hand, since we work in the random oracle model for the signature, we can replace the commitment scheme Com of the Protocol 5 by a single hash function Hash_0. We can then avoid sampling a commitment randomness and hence we can merge the seeds $\mathsf{seed}_i^{[j]}$ and $\mathsf{sseed}_i^{[j]}$.

Finally, to avoid that seed collisions produce the commitment collisions in distinct signatures, we add more entropy by using a *salt* as suggested in [Cha22].

5.2 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. $(\mathsf{seed}_H, x) \leftarrow \mathrm{PRG}(\mathsf{mseed})$ where x is sampled in $\{x \in \mathbb{F}_2^m \mid \mathrm{wt}(x) = w\}$
3. $H \leftarrow \mathrm{PRG}(\mathsf{seed}_H)$
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 exposition below and assume $pk = (H, y)$ and $sk = (H, y, x)$.

The signature algorithm. Given a secret key $sk = (H, y, x)$ and a message $m \in \{0,1\}^*$, the algorithm Sign proceeds as follows:

Step 0:

1. Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.
2. Choose uniform $\mathsf{mseed}^{[0]} \in \{0,1\}^{\lambda}$.
3. Compute the seeds $\mathsf{mseed}^{[1]}, \ldots, \mathsf{mseed}^{[M]}$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[0]})$.

Step 1: For each $j \in [M]$:

1. Use $\mathsf{mseed}^{[j]}$ to generate values $\mathsf{seed}_1^{[j]}, \ldots, \mathsf{seed}_n^{[j]}$ and $r^{[j]} \in \mathrm{Ker}(H)$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[j]})$.
2. For $i \in [n]$, sample $\sigma_i^{[j]}, s_i^{[j]}$ using $\mathsf{seed}_i^{[j]}$ and compute $c_i^{[j]} := \mathrm{Hash}_0(\mathsf{salt}, j, i, \mathsf{seed}_i^{[j]})$.
3. (Cut-and-choose phase) Compute

$$\begin{aligned} \sigma^{[j]} &:= \sigma_n^{[j]} \circ \cdots \circ \sigma_1^{[j]} \\ s^{[j]} &:= s_n^{[j]} + \sigma_n^{[j]}(\cdots + \sigma_2^{[j]}(s_1^{[j]})) \\ q^{[j]} &:= \sigma^{[j]}(r^{[j]}) + s^{[j]} \end{aligned}$$

4. (Sound phase) Compute

$$\begin{aligned} v^{[j]} &:= \sigma^{[j]}(x) \\ \tilde{x}^{[j]} &:= x + r^{[j]} \\ u_0^{[j]} &:= \tilde{x}^{[j]} \\ u_i^{[j]} &:= \sigma_i^{[j]}(u_{i-1}^{[j]}) + s_i^{[j]} \text{ for all } i \in [n] \end{aligned}$$

5. Compute $h_j := \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_n^{[j]})$ and $h'_j := \mathrm{Hash}_2(v^{[j]}, \tilde{x}^{[j]}, (u_i^{[j]})_i)$.

Step 2: Compute

$$(J, L) := \mathrm{Hash}'(m, \mathsf{salt}, h_1, \ldots, h_M, h')$$

with $h' := \mathrm{Merkle}(h'_1, \ldots, h'_M)$, where $J \subset [M]$ is a set of size τ and L is a list $\{\ell_j\}_{j \in J}$ with $\ell_j \in [n]$. The signature includes (J, L) and salt.

Step 3: For each $j \in J$, the signer includes $v^{[j]}$, $\tilde{x}^{[j]}$, $c_{\ell_j}^{[j]}$, $u_{\ell_j}^{[j]}$ and $\mathsf{nodes}^{[j]} := \mathsf{nodes}(\mathsf{mseed}^{[j]}, [n] \backslash \{l_j\})$, where $\mathsf{nodes}()$ returns the minimal nodes which enable to rebuild the tree leaves at input indices in the generation tree with input master seed (see Section 2). Also, the signer includes $\mathsf{nodes}^{\mathrm{M}} := \mathsf{nodes}(\mathsf{mseed}^{[0]}, [M] \backslash J)$ and $\mathsf{auth}^{\mathrm{Merkle}} := \mathsf{auth}((h'_1, \ldots, h'_M), J)$, where $\mathsf{auth}(\cdot)$ returns the Merkle nodes needed to open the paths for the leaves at indices $i \in J$ in the corresponding Merkle tree (see Section 2).

The verification algorithm. Given a public key $pk = (H, y)$, a signature s and a message $m \in \{0,1\}^*$, the algorithm Verif proceeds as follows:

Step 0: Parse the signature s as

$$(\mathsf{salt}, J, L, \mathsf{nodes}^{\mathrm{M}}, \mathsf{auth}^{\mathrm{Merkle}}, \{v^{[j]}, \tilde{x}^{[j]}, \mathsf{nodes}^{[j]}, c_{\ell_j}^{[j]}, u_{\ell_j}^{[j]}\}_{j \in J}) \ .$$

Step 1: Use $\mathsf{nodes}^{\mathrm{M}}$ to rebuild $\mathsf{mseed}^{[j]}$ for each $j \notin J$. Then for every $j \notin J$, use $\mathsf{mseed}^{[j]}$ to compute h_j as in the signature algorithm.

Step 2: For every $j \in J$:

1. Use $\mathsf{nodes}^{[j]}$ to rebuild $\mathsf{seed}_i^{[j]}$ for each $i \neq \ell_j$.
2. For $i \neq \ell_j$, set $c_i^{[j]} := \mathrm{Hash}_0(\mathsf{salt}, j, i, \mathsf{seed}_i^{[j]})$ and get $\sigma_i^{[j]}, s_i^{[j]}$ using $\mathsf{seed}_i^{[j]}$.
3. Compute

$$\begin{aligned} u_i^{[j]} &= \sigma_i^{[j]}(u_{i-1}^{[j]}) + s_i^{[j]} \text{ for all } i \neq \ell_j \\ q^{[j]} &= u_n^{[j]} - v^{[j]} \end{aligned}$$

4. Compute $h_j := \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_n^{[j]})$.
5. Compute $h'_j := \mathrm{Hash}_2(v^{[j]}, \tilde{x}^{[j]}, (u_i^{[j]})_i)$.
6. Check $H\tilde{x}^{[j]} = y$.
7. Check $\mathrm{wt}(v^{[j]}) = w$.

Step 3: Rebuild h' as $\mathrm{Merkle}(h'_1, \ldots, h'_M)$ using $\{h'_j\}_{j\in J}$ and $\mathsf{auth}^{\mathrm{Merkle}}$. Then check that (J, L) equals $\mathrm{Hash}'(m, \mathsf{salt}, h_1, \ldots, h_M, h')$.

5.3 Security Proof

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

Theorem 7. *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 syndrome decoding problem. Model* Hash_0, Hash_1, Hash_2 *and* Hash' *as random oracles where* Hash_0, Hash_1, Hash_2 *have 2λ-bit output length. Then any adaptive chosen-message adversary, running in time t, making q_s signing queries, and making q_0, q_1, q_2, q' queries, respectively, to the random oracles, succeeds in producing a valid forgery with probability upper bounded as*

$$p_{forge} \leq O(q_s \cdot \tau \cdot \varepsilon_{PRG}) + O\left(\frac{(q_0 + q_1 + q_2 + Mnq_s)^2}{2^\lambda}\right) + \varepsilon_{SD} + q' \cdot \varepsilon(M, n, \tau),$$

where

$$\varepsilon(M, n, \tau) := \max_{M-\tau \leq k \leq M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot n^{k-M+\tau}} \right\}.$$

6 Performances

6.1 Communication Cost and Signature Size

In the following analysis, we exclude the challenges from the communication cost since they are of very moderate impact and they does not appear in the signature. The communication then consists into

- $\textsc{Com} := h$,
- $\textsc{Res}_1 := \left((\mathsf{mseed}^{[j]})_{j\notin J}, h', (v^{[j]}, \tilde{x}^{[j]})_{j\in J}\right)$ and
- $\textsc{Res}_2 := \left((\mathsf{seed}_i^{[j]})_{i\neq \ell_j},\ u_{\ell_j}^{[j]},\ c_{\ell_j}^{[j]}\right)_{j\in J}$.

Whereas $u_{\ell_j}^{[j]}$ are full vectors of $\mathbb{F}_2^m$, $v^{[j]}$ are only vectors of weight w and $\tilde{x}^{[j]}$ are vectors from $\{\tilde{x} \mid H\tilde{x} = y\}$, which can be respectively encoded in $\log_2 \binom{m}{w}$ bits and k bits.

Thanks to the use of PRG trees, the communication cost for master seeds is at most $\tau \cdot \log_2 \frac{M}{\tau} \cdot \lambda$ bits (for $\tau \cdot \log_2 \frac{M}{\tau}$ intermediate tree seeds instead of $M - \tau$ leaf seeds) and the communication cost for subseeds is $\log_2(n) \cdot \lambda$ bits (for $\log_2(n)$ intermediate tree seeds instead of $n - 1$ leaf seeds).

So, the total communication cost (in bits) of the protocol is

$$\begin{aligned}\mathrm{Cost} &= \mathrm{Cost}_{\textsc{Com}} + \mathrm{Cost}_{\textsc{Res}_1} + \mathrm{Cost}_{\textsc{Res}_2} \\ &= 4\lambda + \lambda \cdot \tau \cdot \log_2 \frac{M}{\tau} + \tau \cdot \left[2\lambda + (m + k) + \log_2 \binom{m}{w} + \lambda \cdot \log_2(n)\right].\end{aligned}$$

The signature is composed of the same elements, except on two points:

- We need to add the sibling paths of $\{h'_j\}_{j\in J}$ in the Merkle tree to be able to rebuild h'. The communication cost for those paths is at most $(2\lambda) \cdot \tau \cdot \log_2 \frac{M}{\tau}$ bits (for $\tau \cdot \log_2 \frac{M}{\tau}$ intermediate tree labels).

– Since both challenges are merged, there is a single hash value to represent them. And using it and the other components in the signature, it is possible to deduce h and h'.

Thus the signature size (in bits) is

$$\text{Size} = 2\lambda + (3\lambda) \cdot \tau \cdot \log_2 \frac{M}{\tau} + \tau \cdot \left[2\lambda + (m+k) + \log_2 \binom{m}{w} + \lambda \cdot \log_2(n) \right].$$

6.2 Choice of the SD Parameters

In order to simplify the analysis, we place ourselves in a parameter range where the Hamming weight of the secret solution x is close but slightly below the Gilbert-Varshamov bound. This ensures the unicity of the solution with high probability while increasing the difficulty of finding solutions. Furthermore, this choice prevents the applicability of GBA methods. As a consequence, we only need to estimate the complexity of ISD algorithms. Previous studies of such parameters such as [TS16,BBC+19] have argued that the algorithm of May, Meurer and Thomae [MMT11] is the most practical choice in the cryptographic range, as it outperforms algorithms with better asymptotics, e.g. the algorithm from [BJMM12]. Since the algorithm is quite sophisticated and involves several levels of recursion, we simplify the analysis by only considering the cost of the ISD loop, the size of lists at the top-level of their matching technique and the cost of merging these lists. More precisely, with m, k and w being given as input we choose two parameters ℓ and p and compute the following lower bound on the complexity of the attack:

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

Optimizing for ℓ and p yields a slightly conservative estimate for the security level, which we used while choosing our parameters. Given these considerations, we suggest the following concrete parameters to achieve $\lambda \in \{128, 192, 256\}$ bits of security:

– for $\lambda = 128$: $(m = 1280\,,\ k = n/2\,,\ w = 132)$
– for $\lambda = 192$: $(m = 1920\,,\ k = n/2\,,\ w = 200)$
– for $\lambda = 256$: $(m = 2432\,,\ k = n/2\,,\ w = 258)$

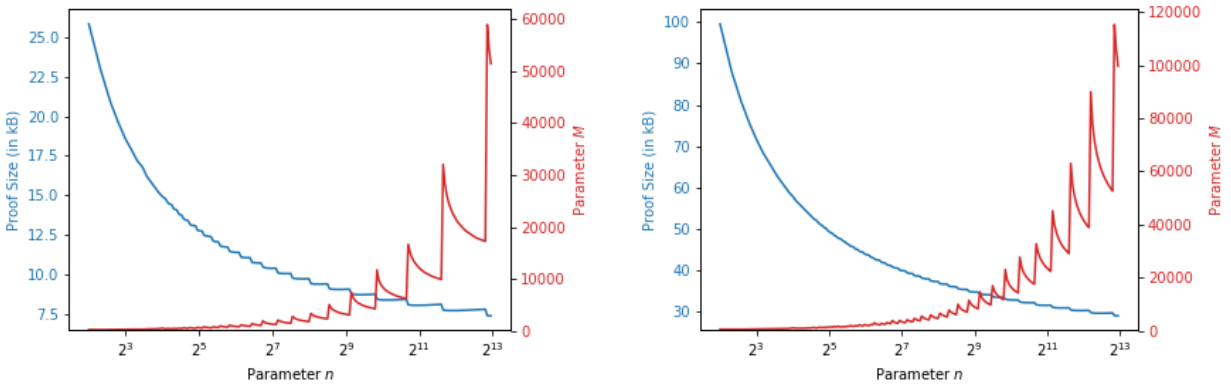
6.3 Impact of the Other Parameters

We have fixed the SD parameters, we now need to choose the remaining parameters: n, M and τ. For a given n, the best proof size is achieved by minimizing τ, and then we take the minimal possible value for M. This is illustrated by Figure 2.

Figure 2 gives the proof size and the required parameter M w.r.t. the parameter n for 128-bit and 256-bit security. We observe that the value of M explodes when we try to achieve very compact signature by increasing n. Since M has a direct impact on the computation time, this parameter provides a trade-off between proof size and computation.

We implemented the signature scheme in C. To sample the permutations, we choose the Fisher–Yates approach where we sample numbers by groups in order to minimize the pseudo-randomness consumption and the rejection rate. 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 which supports AVX2 and AES instructions. All the reported timings were measured on this CPU while disabling Intel Turbo Boost.

We instantiate two trade-offs per security level: 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 2. We provide the measured computational performances of our signature implementation in Table 3.



(a) For 128-bit security

(b) For 256-bit security

Fig. 2: Best proof size (and the used M) according to the parameter n.

The overhead in the computation of our implementation is the uniform sampling (and its application) of the $M \cdot n$ permutations. We tried to implement them as efficiently as possible without introducing a bias using AVX instructions set, but it remains quite expensive. A way to improving the performances of the scheme would be to have a very efficient algorithm for random permutations (for example, by restricting the set in which the permutations are sampled).

λ	Aim	Parameters			$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{sk}\rvert$	ZK Protocol	Signature	
		n	M	τ			$\lvert\mathsf{proof}\rvert$ (max)	$\lvert\mathsf{sgn}\rvert$ (max)	$\lvert\mathsf{sgn}\rvert$ (avg, std)
128	Fast	8	187	49	96	16	21 232	24 372	23 102, 196
128	Short	32	389	28	96	16	13 952	17 540	16 344, 236
192	Fast	8	283	73	144	24	47 400	54 396	51 635, 364
192	Short	32	578	42	144	24	31 344	39 396	36 639, 434
256	Fast	8	379	97	184	32	80 832	93 220	88 412, 559
256	Short	32	767	56	184	32	53 888	68 196	63 165, 655

Table 2: Parameters (n, M, τ) with the achieved communication costs (in bytes). The average signature size and standard deviation has been measured over 10 000 experiments.

7 Comparison

In this section, we compare our scheme to different code-based and post-quantum signature schemes. We start with an analysis of generic zero-knowledge techniques based on MPC-in-the-head paradigm when applied to the syndrome decoding problem. We then compare our scheme with code-based signatures from the literature, as well as other post-quantum signature schemes.

7.1 Comparison with Generic MPC-in-the-Head Techniques

To build a zero-knowledge protocol for the syndrome decoding problem, a possible approach consists in applying generic techniques based on the MPC-in-the-head paradigm [IKOS07], and in particular the efficient scheme due to Katz, Kolesnikov and Wang [KKW18] (on which relies the last version of the Picnic signature

λ	Aim	Keygen	Sign	Verify
128	Fast	0.02 ms 83 632 cycles	12.9 ms 50 641 201 cycles	12.2 ms 47 191 119 cycles
128	Short	0.02 ms 82 990 cycles	62.3 ms 248 805 564 cycles	56.6 ms 220 959 117 cycles
192	Fast	0.04 ms 140 704 cycles	33.9 ms 132 963 552 cycles	32.5 ms 125 421 336 cycles
192	Short	0.04 ms 140 558 cycles	142.9 ms 569 571 538 cycles	125.6 ms 492 359 196 cycles
256	Fast	0.06 ms 225 112 cycles	64.2 ms 248 736 806 cycles	62.7 ms 240 925 977 cycles
256	Short	0.06 ms 224 560 cycles	259.4 ms 1 029 393 486 cycles	229.9 ms 895 729 132 cycles

Table 3: Benchmarks of our signature implementation. Timings are the averaged over 10 000 measurements. The CPU clock cycles have been measured using SUPERCOP (`https://bench.cr.yp.to/supercop.html`).

scheme). Such techniques give a way to build a zero-knowledge protocol proving, for a given boolean circuit C, that the prover knows a witness x such that $C(x) = 1$. The communication cost of the resulting protocol depends mainly on the number of AND gates in the circuit C. To get a zero-knowledge proof for the syndrome decoding problem, we can simply apply theses techniques to a circuit $C_{H,y}$ which on input $x \in \mathbb{F}_2^m$ computes

$$C_{H,y}(x) = \begin{cases} 1 \text{ if } Hx = y \text{ and } \mathrm{wt}(x) = w \\ 0 \text{ otherwise} \end{cases} .$$

We describe a natural approach to build such a circuit while trying to minimize the number $|C_{H,y}|$ of AND gates in supplementary material. The obtained values for $|C_{H,y}|$ are summarized in n Table 4. Applying the KKW scheme [KKW18], the obtained communication cost (in bits) is given by

$$4\lambda + \lambda \cdot \tau \cdot \log_2 \frac{M}{\tau} + \tau \cdot \left(2|C_{H,y}| + m + \lambda \cdot \log_2(n) + 2\lambda \right) .$$

As for our construction, the communication cost depends on the chosen value n, M, τ. In Table 4, the obtained communication cost is given for each set of syndrome decoding parameters. We observe that a simple application of generic MPC-in-the-head techniques on the syndrome decoding problem results in a communication cost which is significantly heavier than with our construction.

Security	m	k	w	$\lvert C_{H,y}\rvert$	Best size for $n = 8$	Best size for $n = 32$
128	1280	640	132	≈ 3800	≈ 52 KB	≈ 33 KB
192	1920	960	200	≈ 5300	≈ 110 KB	≈ 71 KB
256	2432	1216	258	≈ 6100	≈ 175 KB	≈ 112 KB

Table 4: Communication costs of a pure-MPC approach.

Remark 4. There exists various schemes which convert a boolean circuit into a zero-knowledge protocol using the MPC-in-the-head paradigm, but for the range of 300-100000 AND gates, the protocol described in [KKW18] has the smallest communication cost.

7.2 Comparison with Other Code-Based Signature Schemes

In the state of art, there exists two types of signatures. On one hand, there are schemes based on the Fiat-Shamir transform of identification schemes. However, classical approaches to produce identification schemes from code-based problems, like the famous Stern protocol, give large signatures because of the large 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 for which the 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, the authors of [GPS22] propose a zero-knowledge protocol with better soundness using the principle of *MPC-in-the-Head with preprocessing* [KKW18,Beu20] and relying on the syndrome decoding problem on larger fields. On the other hand, the hash-and-sign paradigm using trapdoors is also a popular way to derive signature schemes. Wave is such a recent code-based signature scheme in this paradigm [DST19]. However, such schemes are often more vulnerable to structural attacks.

Scheme Name	Year	$\lvert\mathsf{sgn}\rvert$	$\lvert\mathsf{pk}\rvert$	t_{sgn}	t_{verif}
Stern	1993	62.5 KB	0.09 KB	-	-
Wave	2019	2.07 KB	3.2 MB	300 ms	-
Durandal - I	2018	3.97 KB	14.9 KB	4 ms	5 ms
Durandal - II	2018	4.90 KB	18.2 KB	5 ms	6 ms
LESS-FM - I	2020	15.2 KB	9.77 KB	-	-
LESS-FM - II	2020	5.25 KB	206 KB	-	-
LESS-FM - III	2020	10.39 KB	11.57 KB	-	-
[GPS22]-256	2021	22.2 KB	0.11 KB	-	-
[GPS22]-1024	2021	19.5 KB	0.12 KB	-	-
Our scheme (fast)	2021	22.6 KB	0.09 KB	13 ms	12 ms
Our scheme (short)	2021	16.0 KB	0.09 KB	62 ms	57 ms

Table 5: Comparison of our scheme with signatures from the literature (128-bit security). The performance of Stern protocol is computed for the same SD parameters as us. Reported timings are from the original publications: Wave has been benchmarked on a 3.5 Ghz Intel Xeon E3-1240 v5, while Durandal on a 2.8 Ghz Intel Core i5-7440HQ.

Our proposal is a signature scheme built from a zero-knowledge identification protocol with arbitrary soundness error. In Table 5, we compare the performances of our scheme with the current code-based signature state of the art, for the 128-bit security level.

Our scheme is comparable to Durandal, LESS and [GPS22] for the $\lvert\mathsf{sgn}\rvert + \lvert\mathsf{pk}\rvert$ metric, and much lighter than Wave which features heavy public keys. Regardless of the key size, Wave currently achieves the shortest signatures (but has a slow signing time). In terms of security, our scheme has the advantage of relying on the hardness of one of the oldest problem of the code-based cryptography: the syndrome decoding of random linear codes in Hamming weight metric. Previous schemes based on this problem are obtained from the Stern identification protocol (and its variants). As we observe from Table 5 signatures obtained with our scheme are three times shorter. [GPS22] also relies on the syndrome decoding problem. The technique they use gives a protocol for which the soundness error depends on the size of the base field of the syndrome decoding instance. For this reason, the underlying field must be large enough in order to achieve practical signature sizes. In contrast, the soundness error of our construction is independent of the base field and we can achieve practical signature sizes with the binary field. Let us remark that it would be possible to generalize our protocol to larger fields as well (we simply need to use isometries as in [GPS22] instead of permutations), but this would tend to increase the size of the obtained signatures.

7.3 Comparison with other Post-Quantum Signature Schemes

Finally, we compare in Table 6 our construction with some signature schemes aiming at post-quantum security and which are based from symmetric cryptography primitives (either based on hash tree or on the MPC-in-the-Head paradigm).

Scheme Name	$\|\mathsf{sgn}\|$	t_{sgn}	t_{verif}
Our scheme (short)	16.0 K	62	57
Our scheme (fast)	22.6 K	13	12
SPHINCS$^+$-128s	7.7 K	239	0.7
SPHINCS$^+$-128f	16.7 K	14	1.7
Picnic3	12.3 K	5.2	4.0
BBQ	31.6 K	-	-
Banquet (short)	13.0 K	44	40
Banquet (fast)	19.3 K	6	5

(a) For 128-bit security

Scheme Name	$\|\mathsf{sgn}\|$	t_{sgn}	t_{verif}
Our scheme (short)	61.7 K	259	230
Our scheme (fast)	86.3 K	64	62
SPHINCS$^+$-256s	29.1 K	310	1.5
SPHINCS$^+$-256f	48.7 K	39	2.9
Picnic3	47.6 K	18	13
BBQ	133.7 K	-	-
Banquet (short)	52.8 K	191	175
Banquet (fast)	81.5 K	28	22

(b) For 256-bit security

Table 6: Comparison of our scheme with some post-quantum signature based on symmetric cryptography primitives. The sizes are in bytes and the timings are in milliseconds. The benchmarks for the other schemes have been obtained on an Intel Xeon W-2133 CPU at 3.60GHz. the values for SPHINCS$^+$ and Banquet are extracted from the Banquet article [BdK$^+$21] and the values for Picnic3 are extracted from its original article [KZ20b].

All these schemes have short public and private keys (all under 100 bytes for 128-bit security), which is why we omit the key sizes in the comparison table. Compared to our scheme, Picnic3 [KZ20b] –which also relies on the KKW scheme [KKW18]– has better performances. On the other hand, our scheme is arguably more conservative in terms of security since Picnic is based on the hardness of inverting LowMC [ARS$^+$15], a cipher with unconventional design choices, while our scheme is based on the hardness of the syndrome decoding problem on linear codes, which has a long cryptanalysis history and is believed to be very robust. BBQ [dDOS19] and Banquet [BdK$^+$21] are two other schemes based on the MPC-in-the-Head paradigm and for which the security is based on the hardness of inverting AES (instead of LowMC) which is a more conservative choice. Our signature has better performances than BBQ and is comparable to Banquet. In contrast, it is not competitive with SPHINCS$^+$ [BHK$^+$19] which can achieve shorter signature sizes and very efficient verification. Let us stress that our implementation is a proof of concept which has not been deeply optimized and that some speed-up could probably be obtained by a thorough implementation study (in particular for the sampling and application of permutations). This issue is let open to further research.

Acknowledgements. This work has been supported by the European Union's H2020 Programme under grant agreement number ERC-669891.

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 Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 418–433. Springer, Heidelberg, April / May 2002.

ABG+19. Nicolas Aragon, Olivier Blazy, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. Durandal: A rank metric based signature scheme. In Yuval Ishai and Vincent Rijmen, editors, *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 Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar R. Weippl, and Lida Xu, editors, *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 Elisabeth Oswald and Marc Fischlin, editors, *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-François Biasse, Edoardo Persichetti, and Paolo Santini. LESS-FM: Fine-tuning signatures from the code equivalence problem. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, pages 23–43. Springer, Heidelberg, 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 Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 266–297. Springer, Heidelberg, May 2021.

Beu20. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 183–211. Springer, Heidelberg, May 2020.

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

BJMM12. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, Heidelberg, April 2012.

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

CDG+20. 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, 2020. `https://raw.githubusercontent.com/microsoft/Picnic/master/spec/design-v2.2.pdf`.

Cha22. André Chailloux. On the (In)security of optimized Stern-like signature schemes. WCC 2022: The Twelfth International Workshop on Coding and Cryptography, 2022. `https://www.wcc2022.uni-rostock.de/storages/uni-rostock/Tagungen/WCC2022/Papers/WCC_2022_paper_54.pdf`.

dDOS19. Cyprien de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. BBQ: Using AES in picnic signatures. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 669–692. Springer, Heidelberg, August 2019.

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 Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 21–51. Springer, Heidelberg, December 2019.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *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.

GPS22. Shay Gueron, Edoardo Persichetti, and Paolo Santini. Designing a practical code-based signature scheme from zero-knowledge proofs with trusted setup. *Cryptography*, 6(1), 2022.

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

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.

KZ20a. Daniel Kales and Greg Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *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. `https://tches.iacr.org/index.php/TCHES/article/view/8680`.

Lyu09. Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Heidelberg, December 2009.

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *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.

Sch90. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

Ste94. Jacques Stern. A new identification scheme based on syndrome decoding. In Douglas R. Stinson, editor, *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 Tsuyoshi Takagi, editor, *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.

A Splitting Lemma

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

Lemma 1 (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. $\Pr\left[(x,y) \in B \mid (x,y) \leftarrow A\right] \geq \alpha$.

B Proof of Theorem 2 (Zero Knowledge of Protocol 2)

Proof. We first describe a way to generate an identical distribution to the internal variables of Protocol 2 and then describe the zero-knowledge simulator.

Identical distribution One way to generate an identical distribution to that of variables of Protocol 2 is as follow:

1. Sample $\tilde{x} \leftarrow \{\tilde{x} \mid H\tilde{x} = y\}$.
2. Sample $q \leftarrow \mathbb{F}_2^m$.
3. Sample $v \leftarrow \{v \in \mathbb{F}_2^m \mid \mathrm{wt}(v) = w\}$.
4. For every $i \in [n] \setminus \{i^*\}$, sample $A_i \equiv (\sigma_i, s_i)$ uniformly.
5. For $i \in \{1, \ldots, i^* - 1\}$:
   compute $u_i := A_i(u_{i-1})$ with $u_0 := \tilde{x}$.
6. For i from $n-1$ down to i^*:
   compute $u_i := A_{i+1}^{-1}(u_{i+1})$ with $u_n := v + q$.
7. Randomly sample $A_{i^*} \equiv (\sigma_{i^*}, s_{i^*})$ such that:
   - $u_i = A_{i^*}(u_{i^*-1})$
   - $\alpha = \sigma_{i^*}(\beta)$ with
   $$\begin{cases} \alpha := \sigma_{i^*+1}^{-1} \circ \cdots \circ \sigma_n^{-1}(v) \\ \beta := \sigma_{i^*-1} \circ \cdots \circ \sigma_1(x) \end{cases}.$$

Zero-Knowlegde Simulator The simulator $\mathcal{S}$ proceeds as follows:

- It first samples a random challenge i^* from $\{1, \ldots, n\}$;
- It then performs steps 1 to 6 of the above description and computes commitments c_i's for the (σ_i, s_i)'s, for all $i \neq i^*$;
- Note that the simulator cannot perform step 7 because it requires the knowledge of x (to get β). Instead, the simulator computes a commitment c_{i^*} for a random pair (σ_{i^*}, s_{i^*});
- The simulator calls $\tilde{\mathcal{V}}$ with $\textsc{Com} := (\{c_i\}_i, q, v, \tilde{x}, \{u_i\}_i)$ and, restart the simulation from scratch if $\tilde{\mathcal{V}}$ does not return i^*, and output the simulated transcript otherwise.

The output transcript is independent and identically to the genuine transcript except for c_{i^*}. Distinguishing then means breaking the commitment hiding property.

□

C Proof of Theorem 3 (Soundness of Protocol 2)

Proof. Let us first show how to extract the syndrome decoding solution x from a few transcripts satisfying specific conditions. We will then show how to get such transcripts from a rewindable black-box access to $\tilde{\mathcal{P}}$.

Transcripts used for extraction. We assume that we can extract two transcripts

$$\begin{aligned} T_1 &:= (\{c_i\}_i, q^\star, v, \tilde{x}, \{u_i\}_i, \mathrm{CH}_1 := i_1^*, (\sigma_i, s_i, \rho_i)_{i \neq i_1^*}) \\ T_2 &:= (\{c_i\}_i, q^\star, v, \tilde{x}, \{u_i\}_i, \mathrm{CH}_2 := i_2^*, (\sigma'_i, s'_i, \rho'_i)_{i \neq i_2^*}) \ . \end{aligned}$$

Using these two transcripts, we next show that it is possible to extract a solution of the syndrome decoding instance defined by H and y. We can assume that the $(\sigma_i, s_i)_{i \notin \{i_1^*, i_2^*\}}$ and $(\sigma'_i, s'_i)_{i \notin \{i_1^*, i_2^*\}}$ are mutually consistent between the two transcripts, since otherwise we find a commitment collision for at least one of the commitments $\{c_i\}_i$. So, we know $(\sigma_i^{[j_0]}, s_i^{[j_0]})$ for all $i \in \{1, \ldots, n\}$ from T_1 and T_2. We define $(\sigma_{i_1^*}, s_{i_1^*}) := (\sigma'_{i_1^*}, s'_{i_1^*})$.

Extraction of x from T_1 and T_2. In the following, we will denote $\mathcal{V}_{\mathrm{CH}_1}$ (resp. $\mathcal{V}_{\mathrm{CH}_2}$) the set of checked equations at the end of the transcript with CH_1 (resp. CH_2) as challenge.

Let us define $\sigma := \sigma_n \circ \cdots \circ \sigma_1$ and $x' := \sigma^{-1}(v)$. We simply return x' as a candidate solution for x. Because $\mathrm{wt}(v) = w$ (from $\mathcal{V}_{\mathrm{CH}_1}$ or $\mathcal{V}_{\mathrm{CH}_2}$), we have $\mathrm{wt}(x') = w$. We now show that we further have $y = Hx'$.

Thanks to $\mathcal{V}_{\mathrm{CH}_1}$, we know that

$$\forall i \in [n] \backslash \{i_1^*\}, u_i = \sigma_i(u_{i-1}) + s_i.$$

And thanks to $\mathcal{V}_{\mathrm{CH}_2}$, we get the remaining equation

$$u_{i_1^*} = \sigma_{i_1^*}(u_{i_1^*-1}) + s_{i_1^*}.$$

So, we know that

$$\begin{aligned} u_n &= s_n + \sigma_n(s_{n-1} + \sigma_{n-1}(\ldots + \sigma_2(s_1 + \sigma_1(u_0)))) \\ &= s_n + \sigma_n(s_{n-1} + \sigma_{n-1}(\ldots + \sigma_2(s_1 + \sigma_1(\tilde{x})))) \\ &= \underbrace{(\sigma_n \circ \cdots \circ \sigma_1)}_{\sigma}(\tilde{x}) + \underbrace{s_n + \sigma_n(s_{n-1} + \sigma_{n-1}(\ldots + \sigma_2(s_1)))}_{s} \\ &= \sigma(\tilde{x}) + s \end{aligned} \tag{6}$$

Now, we have

$$\begin{aligned} Hx' &= H\sigma^{-1}(v) \\ &= H\sigma^{-1}(u_n - q) && \text{from } \mathcal{V}_{\mathrm{CH}_1} \text{ or } \mathcal{V}_{\mathrm{CH}_2} \\ &= H\sigma^{-1}(\sigma(\tilde{x}) + s - q) && \text{from 6} \\ &= H\tilde{x} - H\sigma^{-1}(q - s) \\ &= y - H\sigma^{-1}(q - s) && \text{from } \mathcal{V}_{\mathrm{CH}_1} \text{ or } \mathcal{V}_{\mathrm{CH}_2} \end{aligned}$$

Since q has been honestly built and $\{(\sigma_i, s_i)\}_i$ has been extracted from $\tilde{\mathcal{P}}$, we know there exists $r \in \mathrm{Ker}(H)$ such that

$$q = \sigma(r) + s.$$

And so,

$$Hx' = y - H\sigma^{-1}(\sigma(r)) = y - Hr = y.$$

So, we well obtain $Hx' = y - H\sigma^{-1}(\sigma(r)) = y - Hr = y$.

Extraction of T_1 and T_2 from $\tilde{\mathcal{P}}$. Let now show how to extract these two transcripts from $\tilde{\mathcal{P}}$. We want these transcripts to be with the same commitment COM from the prover but with different challenges. We define the following extractor $\mathcal{E}$.

Extractor $\mathcal{E}$:

1. Repeat $+\infty$ times:
2. Run $\tilde{\mathcal{P}}$ with honest $\mathcal{V}$ to get transcript T
3. If T_0 is not a successful transcript, go to the next iteration
4. Do N_1 times:
5. Run $\tilde{\mathcal{P}}$ with honest $\mathcal{V}$ and same r_{COM} as for T to get T'
6. If T is a successful transcript s.t. its challenge is not the same than for T_0, return (T, T')

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 $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}] = \tilde{\varepsilon}$. We shall further denote by R_{COM} the randomness of $\tilde{\mathcal{P}}$ that is used to generate the initial commitment $\text{COM} = (\{c_i\}_i, q^\star, v, \tilde{x}, \{u_i\}_i)$.

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_{COM} be a possible realisation of R_{COM}. We will say that r_{COM} is *good* if

$$\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_{\text{COM}} = r_{\text{COM}}] \geq (1-\alpha)\tilde{\varepsilon}. \tag{7}$$

By the Splitting Lemma 1 (see Appendix A) we have

$$\Pr[R_{\text{COM}} \text{ is good} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}] \geq \alpha \ . \tag{8}$$

Let us define the *collision event* of T and T' as the event

$$\mathsf{Col}(T, T') := \text{“}T \text{ and } T' \text{ has the same challenge CH”} \ .$$

When T is fixed and T' is random, the collision event occurs with probability

$$p_{col} := \Pr[\mathsf{Col}(T, T')] = \frac{1}{n} = \varepsilon \ .$$

Let us lower bound the probability that an iteration of the inner loop find a right couple (T, T') when R_{COM} is good:

$$p := \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T'} \cap \neg\mathsf{Col}(T, T') \mid R_{\text{COM}} \text{ good}] \ .$$

We have

$$\begin{aligned}
p &= \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T'} \mid R_{\text{COM}} \text{ good}] - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T'} \cap \mathsf{Col}(T, T') \mid R_{\text{COM}} \text{ good}] \\
&\geq (1-\alpha)\cdot\tilde{\varepsilon} - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T'} \cap \mathsf{Col}(T, T') \mid R_{\text{COM}} \text{ good}] && \text{by 7} \\
&\geq (1-\alpha)\cdot\tilde{\varepsilon} - \Pr[\mathsf{Col}(T, T') \mid R_{\text{COM}} \text{ good}] \\
&= (1-\alpha)\cdot\tilde{\varepsilon} - \Pr[\mathsf{Col}(T, T')] && \text{by independence} \\
&= (1-\alpha)\cdot\tilde{\varepsilon} - \varepsilon > 0
\end{aligned}$$

Let define $p_0 := (1-\alpha)\cdot\tilde{\varepsilon} - \varepsilon$. By running $\tilde{\mathcal{P}}$ with the same r_{COM} as for the good transcript N_1 times, we hence obtain a second non-colliding transcript T' with probability at least $1/2$ when

$$N_1 \approx \frac{\ln(2)}{\ln\left(\frac{1}{1-p_0}\right)} \leq \frac{\ln(2)}{p_0} \ . \tag{9}$$

Without assumption on R_{COM}, the probability to find a couple when T is successful satisfies:

$$\Pr[\text{found} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T}] \geq \Pr[R_{\text{COM}} \text{ good} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T}] \cdot \Pr[\text{found} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T} \cap R_{\text{COM}} \text{ good}] \geq \frac{\alpha}{2} \ .$$

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,
- if T is not successful, which occurs with probability $(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T}])$,
  - the extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$,
- if T is successful, which occurs with probability $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T}]$,
  - the extractor makes at most N_1 calls to $\tilde{\mathcal{P}}$ in the inner loop of $\mathcal{E}$,
  - then $\mathcal{E}$ quits the inner loop without returning a couple of transcripts, which occurs with probability $\Pr[\text{not found} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T}]$, the extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$,
  - otherwise, if the inner loop returns a non-empty list, the extractor stops and no more calls to $\tilde{\mathcal{P}}$ are necessary.

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

$$\mathbb{E}[C] = 1 + \underbrace{(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T}]) \cdot \mathbb{E}[C]}_{T \text{ unsuccessful}} + \underbrace{\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T}] \cdot \left(N_1 + \Pr[\text{not found} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T}] \cdot \mathbb{E}[C]\right)}_{T \text{ successful}}$$

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 as

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

which concludes the proof. □

D Proof of Theorem 5 (HVZK of Protocol 5)

Proof. Let us describe the simulator $\mathcal{S}$. Let us denote (J, L) the input of the simulator. First, $\mathcal{S}$ randomly picks the master seeds $\mathsf{seed}^{[0]}$.

- For $j \in [M] \backslash J$, $\mathcal{S}$ follows honestly the protocol since it does not need to know the secret.
- For $j \in J$, $\mathcal{S}$ uses the same method than the one described in the proof of the Theorem 2 (appendix B):
  - Sample $\tilde{x}^{[j]} \leftarrow \{\tilde{x} \mid H\tilde{x} = y\}$.
  - Sample $q^{[j]} \leftarrow \mathbb{F}_2^m$.
  - Sample $v^{[j]} \leftarrow \{v \in \mathbb{F}_2^m \mid \mathrm{wt}(v) = w\}$.
  - For every $i \in [n] \setminus \{i^*\}$, sample $A_i^{[j]} \equiv (\sigma_i^{[j]}, s_i^{[j]})$ uniformly.
  - For $i \in \{1, \ldots, i^* - 1\}$:
    compute $u_i^{[j]} := A_i^{[j]}(u_{i-1}^{[j]})$ with $u_0^{[j]} := \tilde{x}^{[j]}$.
  - For i from $n-1$ down to i^*:
    compute $u_i^{[j]} := (A_{i+1}^{[j]})^{-1}(u_{i+1}^{[j]})$ with $u_n^{[j]} := v^{[j]} + q^{[j]}$.
  - Compute commitments $c_i^{[j]}$'s for the $(\sigma_i^{[j]}, s_i^{[j]})$'s, for all $i \neq i^*$.
  - Computes a commitment $c_{i^*}^{[j]}$ for a random pair $(\sigma_{i^*}^{[j]}, s_{i^*}^{[j]})$.

The output transcript is independent and identically to the genuine transcript except for the randomness sampling and $c_{i^*}^{[j]}$ when $j \in J$. Distinguishing then means breaking the pseudorandomness property or breaking the commitment hiding property.

□

E Proof of Theorem 6 (Soundness of Protocol 5)

Proof. Let us first show how to extract the syndrome decoding solution x from a few transcripts satisfying specific conditions. We will then show how to get such transcripts from a rewindable black-box access to $\tilde{\mathcal{P}}$.

Transcripts used for extraction. We assume that we can extract three transcripts

$$T_i = (\textsc{Com}^{(i)}, \textsc{Ch}_1^{(i)}, \textsc{Rsp}_1^{(i)}, \textsc{Ch}_2^{(i)}, \textsc{Rsp}_2^{(i)}) \quad \text{for } i \in \{1,2,3\} \ , \tag{10}$$

from $\tilde{\mathcal{P}}$, with $\textsc{Ch}_1^{(i)} := J^{(i)}$, $\textsc{Ch}_2^{(i)} := \{\ell_j^{(i)}\}_{j \in J^{(i)}}$, which satisfy:

1. $\textsc{Com}^{(1)} = \textsc{Com}^{(2)} = \textsc{Com}^{(3)} = h$,
2. there exists $j_0 \in (J^{(1)} \cap J^{(2)}) \setminus J^{(3)}$ s.t. $\ell_{j_0}^{(1)} \neq \ell_{j_0}^{(2)}$
3. T_1 and T_2 are success transcripts (*i.e.* which pass all the tests of $\mathcal{V}$),
4. $\mathsf{seed}^{[j_0]}$ from $\textsc{Rsp}_1^{(3)}$ is consistent with the $(\sigma_i^{[j_0]}, s_i^{[j_0]})$ from T_1 and T_2.

Using these three transcripts, we next show that it is possible to extract a solution of the syndrome decoding instance defined by H and y. We can assume that all the revealed $q^{[j]}$ and $(\sigma_i^{[j]}, s_i^{[j]})$ are mutually consistent between the three transcripts, since otherwise we find a hash collision. So, we know $(\sigma_i^{[j_0]}, s_i^{[j_0]})$ for all $i \in \{1, \dots, n\}$ from T_1 and T_2.

Extraction of x from T_1, T_2 and T_3. For this part, we will only consider the variables of the form $(*)^{[j_0]}$, so we will omit the superscript for the sake of clarity. In the following, we will denote $\mathcal{V}_{T_i}$ the set of checked equations at the end of the protocol with T_i for $i \in \{1,2,3\}$.

Let us define $\sigma := \sigma_n \circ ... \circ \sigma_1$ and $x' := \sigma^{-1}(v)$. We simply return x' as a candidate solution for x. Because $\mathrm{wt}(v) = w$ (from $\mathcal{V}_{T_1}$ or $\mathcal{V}_{T_2}$), we have $\mathrm{wt}(x') = w$. We now show that we further have $y = Hx'$.

Thanks to $\mathcal{V}_{T_1}$, we know that

$$\forall i \in [n] \backslash \{\ell_{j_0}\}, u_i = \sigma_i(u_{i-1}) + s_i \ .$$

And thanks to $\mathcal{V}_{T_2}$, we get the remaining equation

$$u_{j_0} = \sigma_{j_0}(u_{j_0 - 1}) + s_{j_0} \ .$$

So, we know that

$$\begin{aligned} u_n &= s_n + \sigma_n(s_{n-1} + \sigma_{n-1}(... + \sigma_2(s_1 + \sigma_1(u_0)))) \\ &= s_n + \sigma_n(s_{n-1} + \sigma_{n-1}(... + \sigma_2(s_1 + \sigma_1(\tilde{x})))) \\ &= \underbrace{(\sigma_n \circ ... \circ \sigma_1)}_{\sigma}(\tilde{x}) + \underbrace{s_n + \sigma_n(s_{n-1} + \sigma_{n-1}(... + \sigma_2(s_1)))}_{s} \\ &= \sigma(\tilde{x}) + s \ . \end{aligned}$$

Now, we have

$$\begin{aligned} Hx' &= H\sigma^{-1}(v) \\ &= H\sigma^{-1}(u_n - q) && \text{from } \mathcal{V}_{T_1} \text{ or } \mathcal{V}_{T_2} \\ &= H\sigma^{-1}(\sigma(\tilde{x}) + s - q) && \text{from the previous calculus} \\ &= H\tilde{x} - H\sigma^{-1}(q - s) \\ &= y - H\sigma^{-1}(q - s) && \text{from } \mathcal{V}_{T_1} \text{ or } \mathcal{V}_{T_2} \end{aligned}$$

From $\mathcal{V}_{T_3}$, we get that there exists $r \in \mathrm{Ker}(H)$ such that

$$q = \sigma(r) + s \ .$$

So, we well obtain $Hx' = y - H\sigma^{-1}(\sigma(r)) = y - Hr = y$.

Extraction of T_1, T_2 and T_3 from $\tilde{\mathcal{P}}$. 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 $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}] = \tilde{\varepsilon}$. We shall further denote by R_h the randomness of $\tilde{\mathcal{P}}$ which is used to generate the initial commitment $\textsc{Com} = h$.

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

$$\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \geq (1-\alpha) \cdot \tilde{\varepsilon} \ . \tag{11}$$

By the Splitting Lemma 1 (see Appendix A) we have

$$\Pr[R_h \text{ good} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}] \geq \alpha \ . \tag{12}$$

Our extractor first obtains a successful transcript T_0 by running the protocol making calls to $\tilde{\mathcal{P}}$ with honest verifier requests. If this T_0 corresponds to a good r_h, then we can obtain further successful transcripts with "high" probability by rewinding the protocol just after the initial commitment $\textsc{Com} = h$. Based on this assumption, a sub-extractor $\mathcal{E}_0$ will build a list of *successful* transcripts $\mathcal{T}$, all with same initial commitment. For every $T \in \mathcal{T}$, we denote J_T the set J (first challenge) for this transcript, as well as $\bar{J}_T := [M] \setminus J_T$, and $L_T = \{\ell_{T,j}\}_{j \in J_T}$ the set L (second challenge) for this transcript. We further denote

$$\bar{J}(\mathcal{T}) := \bigcup_{T \in \mathcal{T}} \bar{J}_T$$

which is the set of indexes $j \in [M]$ for which $q^{[j]}$ has been opened, as well as

$$C(\mathcal{T}) := \{j \mid \exists\, T, T' \in \mathcal{T} \text{ s.t. } j \in J_T \cap J_{T'} \text{ and } \ell_{T,j} \neq \ell_{T',j}\}$$

which is the set of indexes $j \in [M]$ for which all the $(\sigma_i^{[j]}, s_i^{[j]})_i$ have been revealed.

For a certain number N_1 of iterations, the sub-extractor $\mathcal{E}_0$ tries to feed the list $\mathcal{T}$ until the following stop condition is reached:

$$C(\mathcal{T}) \cap \bar{J}(\mathcal{T}) \neq \emptyset \ .$$

If this condition is reached then we have three transcripts $T_1, T_2, T_3 \in \mathcal{T}$ such that

$$\exists j \in (J_{T_1} \cap J_{T_2}) \cap \bar{J}_{T_3} : \ell_{T_1,j} \neq \ell_{T_2,j}$$

which is what we need to recover x (as explained above). We formally describe the sub-extractor routine in the following pseudocode:

Sub-extractor $\mathcal{E}_0$ (on input a successful transcript T_0):

1. $\mathcal{T} = \{T_0\}$
2. Do N_1 times:
3. Run $\tilde{\mathcal{P}}$ with honest $\mathcal{V}$ and same r_h as T_0 to get T
4. If T is a successful transcript:
5. $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$
6. If $C(\mathcal{T}) \cap \bar{J}(\mathcal{T}) \neq \emptyset$, return $\mathcal{T}$.
7. Return $\emptyset$.

Let us know evaluate the probability that the stop condition is reached in a given number of iteration N_1. In the following, we naturally denote $J(\mathcal{T}) := \bigcup_{T \in \mathcal{T}} J_T$, the set of indexes $j \in [M]$ which have been used in the second phase for at least one transcript $T \in \mathcal{T}$. Note that we always have $C(\mathcal{T}) \subseteq J(\mathcal{T})$.

Consider a loop iteration in $\mathcal{E}_0$ at the beginning of which we have a list $\mathcal{T}$ of successful transcripts such that $C(\mathcal{T}) \cap \bar{J}(\mathcal{T}) = \emptyset$ (*i.e.* the stop condition has not been reached) and a transcript T sampled at Step 3. We consider the three following events:

– the event $E_1 := \{J(\mathcal{T}) \subsetneq J(\mathcal{T} \cup \{T\})\}$ which implies that the set $J(\mathcal{T})$ will be increased at Step 5 when $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$ if T is a successful transcript,

– the event $E_2 := \{C(\mathcal{T}) \subsetneq C(\mathcal{T} \cup \{T\})\}$ which implies that the set $C(\mathcal{T})$ will be increased at Step 5 when $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$ if T is a successful transcript,

– the event $E_3 := \{C(\mathcal{T} \cup \{T\}) \cap \bar{J}(\mathcal{T} \cup \{T\}) \neq \emptyset\}$ which implies that the stop condition will be reached at Step 6 if T is a successful transcript.

The above events are defined with respect to the randomness of the two challenges in T. Note these events do not require that T is a successful transcript.

Let us lower bound the probability to have a good transcript T and one of the three events occuring in the presence of a good R_h:

$$p := \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \cap (E_1 \cup E_2 \cup E_3) \mid R_h \text{ good}] \ .$$

We have:

$$\begin{aligned}
p &= \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}] - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \cap (\bar{E}_1 \cap \bar{E}_2 \cap \bar{E}_3) \mid R_h \text{ good}] \\
&\geq (1-\alpha) \cdot \tilde{\varepsilon} - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \cap (\bar{E}_1 \cap \bar{E}_2 \cap \bar{E}_3) \mid R_h \text{ good}] && \text{by (11)} \\
&\geq (1-\alpha) \cdot \tilde{\varepsilon} - \Pr[\bar{E}_1 \cap \bar{E}_2 \cap \bar{E}_3 \mid R_h \text{ good}] \\
&= (1-\alpha) \cdot \tilde{\varepsilon} - \Pr[\bar{E}_1 \cap \bar{E}_2 \cap \bar{E}_3] && \text{by independence.}
\end{aligned}$$

Now, we have

$$\begin{aligned}
\bar{E}_2 \cap \bar{E}_3 \quad &\Leftrightarrow \quad C(\mathcal{T}) = C(\mathcal{T} \cup \{T\}) \quad \cap \quad C(\mathcal{T} \cup \{T\}) \cap \bar{J}(\mathcal{T} \cup \{T\}) = \emptyset \\
&\Leftrightarrow \quad C(\mathcal{T}) = C(\mathcal{T} \cup \{T\}) \quad \cap \quad C(\mathcal{T}) \cap \bar{J}(\mathcal{T} \cup \{T\}) = \emptyset \\
&\Leftrightarrow \quad C(\mathcal{T}) = C(\mathcal{T} \cup \{T\}) \quad \cap \quad (C(\mathcal{T}) \cap \bar{J}(\mathcal{T})) \cup (C(\mathcal{T}) \cap \bar{J}_T) = \emptyset \\
&\Leftrightarrow \quad C(\mathcal{T}) = C(\mathcal{T} \cup \{T\}) \quad \cap \quad (C(\mathcal{T}) \cap \bar{J}_T) = \emptyset \\
&\Leftrightarrow \quad C(\mathcal{T}) = C(\mathcal{T} \cup \{T\}) \quad \cap \quad C(\mathcal{T}) \subseteq J_T
\end{aligned}$$

which also gives

$$\bar{E}_2 \cap \bar{E}_1 \cap \bar{E}_3 \quad \Leftrightarrow \quad C(\mathcal{T}) = C(\mathcal{T} \cup \{T\}) \quad \cap \quad C(\mathcal{T}) \subseteq J_T \subseteq J(\mathcal{T}) \ .$$

So we can further lower bound p as:

$$\begin{aligned}
p &\geq (1-\alpha) \cdot \tilde{\varepsilon} - \Pr[C(\mathcal{T}) \subseteq J_T \subseteq J(\mathcal{T}) \cap C(\mathcal{T}) = C(\mathcal{T} \cup \{T\})] \\
&= (1-\alpha) \cdot \tilde{\varepsilon} - \Pr[C(\mathcal{T}) \subseteq J_T \subseteq J(\mathcal{T})] \\
&\qquad \times \Pr[C(\mathcal{T}) = C(\mathcal{T} \cup \{T\}) \mid C(\mathcal{T}) \subseteq J_T \subseteq J(\mathcal{T})] \\
&= (1-\alpha) \cdot \tilde{\varepsilon} - \frac{\binom{|J(\mathcal{T})| - |C(\mathcal{T})|}{\tau - |C(\mathcal{T})|}}{\binom{M}{\tau}} \cdot \left(\frac{1}{n}\right)^{\tau - |C(\mathcal{T})|}
\end{aligned}$$

By defining $k := M - |C(\mathcal{T})|$, we have

$$\begin{aligned}
\binom{|J(\mathcal{T})| - |C(\mathcal{T})|}{\tau - |C(\mathcal{T})|} &= \binom{|J(\mathcal{T})| - |C(\mathcal{T})|}{|J(\mathcal{T})| - \tau} \\
&= \binom{k - (M - |J(\mathcal{T})|)}{M - \tau - (M - |J(\mathcal{T})|)} \\
&\leq \binom{k}{M - \tau}
\end{aligned}$$

The first equality holds from $\binom{x}{y} = \binom{x}{x-y}$ (for any $x, y \in \mathbb{N}$) while the second holds by definition of k. The inequality holds from $\binom{x}{y} \geq \binom{x-z}{y-z}$ (for any $x, y, z \in \mathbb{N}$ with $z < x, y$). We hence finally get:

$$p \geq (1-\alpha) \cdot \tilde{\varepsilon} - \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot n^{k-M+\tau}} \geq (1-\alpha) \cdot \tilde{\varepsilon} - \varepsilon$$

To summarize, in the presence of a good R_h, the probability of the event $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap (E_1 \cup E_2 \cup E_3)$ (*i.e.* getting a successful transcript T which yields one the the three events E_1, E_2, or E_3) is lower bounded by $(1-\alpha) \cdot \tilde{\varepsilon} - \varepsilon > 0$. Moreover, the event $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap E_1$ can occur at most $M - \tau$ times, because $J_{T_0} \subseteq J(\mathcal{T}) \subseteq \{1, \ldots, M\}$. And the event $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap E_2$ can occur at most τ times before further implying E_3 (because if $|C(\mathcal{T})| \geq \tau$ then adding one more term to $C(\mathcal{T})$ systematically implies E_3). We deduce that after $M+1$ occurrences of $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap (E_1 \cup E_2 \cup E_3)$, the event E_3 must have occurred at least once.

Let us now define

$$N_1 = \frac{4M}{p_0} \quad \text{with} \quad p_0 := (1-\alpha) \cdot \tilde{\varepsilon} - \varepsilon \ . \tag{13}$$

And let $X \sim \mathcal{B}(N_1, p_0)$ a binomial distributed random variable with parameters (N_1, p_0). The probability that $\mathcal{E}_0$ reaches the stop condition and returns a non-empty list for a successful transcript T_0 with good R_h satisfies:

$$\begin{aligned}
\Pr[\mathcal{E}_0(T_0) \neq \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0} \cap R_h \text{ good}] &\geq \Pr[X > M] \\
&= \Pr\left[\frac{X}{N_1} - p_0 > \frac{M}{N_1} - p_0\right] \\
&= 1 - \Pr\left[\frac{X}{N_1} - p_0 < \frac{M}{N_1} - p_0\right] \\
&= 1 - \Pr\left[\frac{X}{N_1} - p_0 < -\frac{3}{4} p_0\right] \\
&\geq 1 - \Pr\left[\left|\frac{X}{N_1} - p_0\right| > \frac{3}{4} p_0\right] \\
&\geq 1 - \frac{p_0 \cdot (1-p_0)}{N_1 \cdot p_0^2 \cdot \left(\frac{3}{4}\right)^2} \qquad (14) \\
&= 1 - \frac{16}{9} \cdot \frac{1-p_0}{4 \cdot M} = 1 - \frac{4}{9} \cdot \frac{1-p_0}{M} \\
&\geq 1 - \frac{4}{9} \geq \frac{1}{2}
\end{aligned}$$

The inequality (14) holds from the Bienaymé-Techbychev inequality. Thus, using $N_1 = \frac{4M}{p_0}$, the probability to reach the stop condition assuming a good R_h is at least $1/2$. Without assumption on R_h, the probability to reach the stop condition satisfies:

$$\begin{aligned}
&\Pr[\mathcal{E}_0(T_0) \neq \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \\
&\qquad \geq \Pr[R_h \text{ good} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \cdot \Pr[\mathcal{E}_0(T_0) \neq \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0} \cap R_h \text{ good}] \geq \frac{\alpha}{2} \ .
\end{aligned}$$

Let us now describe the complete extractor procedure:

Extractor $\mathcal{E}$:

1. Repeat $+\infty$ times:
2. Run $\tilde{\mathcal{P}}$ with honest $\mathcal{V}$ to get transcript T_0
3. If T_0 is not a successful transcript, go to the next iteration
4. Call $\mathcal{E}_0$ on T_0 to get list of transcripts $\mathcal{T}$
5. If $\mathcal{T} \neq \emptyset$, return $\mathcal{T}$

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_0,
- if T_0 is not successful, which occurs with probability $(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}])$,
  - the extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$,
- if T_0 is successful, which occurs with probability $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}]$,
  - the extractor makes at most N_1 calls to $\tilde{\mathcal{P}}$ in the loop of $\mathcal{E}_0$,
  - then $\mathcal{E}_0$ returns an empty list (the stop condition is not reached), which occurs with probability $\Pr[\mathcal{E}_0(T_0) = \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}]$, the extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$,
  - otherwise, if $\mathcal{E}_0(T_0)$ returns a non-empty list, the extractor stops and no more calls to $\tilde{\mathcal{P}}$ are necessary.

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

$$\mathbb{E}[C] = 1 + \underbrace{(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}]) \cdot \mathbb{E}[C]}_{T_0 \text{ unsuccessful}} + \underbrace{\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \cdot \big(N_1 + \Pr[\mathcal{E}_0(T_0) = \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \cdot \mathbb{E}[C]\big)}_{T_0 \text{ successful}}$$

which gives

$$\begin{aligned}
\mathbb{E}[C] &\leq 1 + (1 - \tilde{\varepsilon}) \cdot \mathbb{E}[C] + \tilde{\varepsilon} \cdot \Big(N_1 + \big(1 - \frac{\alpha}{2}\big) \cdot \mathbb{E}[C]\Big) \\
&\leq 1 + \tilde{\varepsilon} \cdot N_1 + \mathbb{E}[C]\Big(1 - \frac{\tilde{\varepsilon} \cdot \alpha}{2}\Big) \\
&\leq \frac{2}{\alpha \cdot \tilde{\varepsilon}} \cdot (1 + \tilde{\varepsilon} \cdot N_1) \\
&= \frac{2}{\alpha \cdot \tilde{\varepsilon}} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{4 \cdot M}{(1 - \alpha) \cdot \tilde{\varepsilon} - \varepsilon}\right)
\end{aligned}$$

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 as

$$\mathbb{E}[C] \leq \frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{8 \cdot M}{\tilde{\varepsilon} - \varepsilon}\right)$$

which concludes the proof. □

F Security Proof of the Signature Scheme

We give herafter the proof of Theorem 7. This proof is a carbon copy of the security proof of the Picnic signature scheme [CDG+20, Theorem 6.2] with few adaptations to our context.

Proof (Theorem 7). Fix some attacker $\mathcal{A}$. 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, Hash_2 made by $\mathcal{A}$, and let q' denote the number of queries to Hash' 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, Hash_2, and Hash' 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. We are interested in upper-bounding $\Pr_1[\mathrm{Forge}]$.

Experiment 2. We abort the experiment if, during the course of the experiment, a collision in Hash_0, Hash_1, or Hash_2 is found. Suppose $q = \max\{q_0, q_1, q_2\}$, then the number of queries to any oracle throughout the experiment (by either the adversary or the signing algorithm) is at most $(q + Mnq_s)$. Thus,

$$|\Pr_1[\mathrm{Forge}] - \Pr_2[\mathrm{Forge}]| \leq 3 \cdot \frac{(q + Mnq_s)^2}{2 \cdot 2^{2\lambda}}.$$

Experiment 3. The difference with the previous experiment is that, when signing a message m we begin by choosing (J, L) uniformly. Steps 1 and 3 of the signing algorithm are computed as before, but in step 2 we simply set the output of Hash' equal to (J, L). Formally, a signature on a message m is now computed as follows:

Step 0:
- Choose uniform (J, L), where $J \subset [M]$ is a set of size τ, and $L = \{\ell_j\}_{j \in J}$ with $\ell_j \in [n]$.
- Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.
- Choose uniform $\mathsf{mseed}^{[0]} \in \{0,1\}^{\lambda}$.
- Compute the seeds $\mathsf{mseed}^{[1]}, \ldots, \mathsf{mseed}^{[M]}$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[0]})$.

Step 1: For each $j \in [M]$:
1. Use $\mathsf{mseed}^{[j]}$ to generate values $\mathsf{seed}_1^{[j]}, \ldots, \mathsf{seed}_n^{[j]}$ and $r^{[j]} \in \mathrm{Ker}(H)$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[j]})$.
2. For $i \in [n]$, sample $\sigma_i^{[j]}, s_i^{[j]}$ using $\mathsf{seed}_i^{[j]}$ and compute $c_i^{[j]} := \mathrm{Hash}_0(\mathsf{salt}, j, i, \mathsf{seed}_i^{[j]})$.
3. (Cut-and-choose phase) Compute

$$\begin{aligned} \sigma^{[j]} &:= \sigma_n^{[j]} \circ \cdots \circ \sigma_1^{[j]} \\ s^{[j]} &:= s_n^{[j]} + \sigma_n^{[j]}(\cdots + \sigma_2^{[j]}(s_1^{[j]})) \\ q^{[j]} &:= \sigma^{[j]}(r^{[j]}) + s^{[j]} \end{aligned}$$

4. (Sound phase) Compute

$$\begin{aligned} v^{[j]} &:= \sigma^{[j]}(x) \\ \tilde{x}^{[j]} &:= x + r^{[j]} \\ u_0^{[j]} &:= \tilde{x}^{[j]} \\ u_i^{[j]} &:= \sigma_i^{[j]}(u_{i-1}^{[j]}) + s_i^{[j]} \text{ for all } i \in [n] \end{aligned}$$

5. Compute $h_j := \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_n^{[j]})$ and $h'_j := \mathrm{Hash}_2(v^{[j]}, \tilde{x}^{[j]}, (u_i^{[j]})_i)$.

Step 2: Set $\mathrm{Hash}'(m, \mathsf{salt}, h_1, \ldots, h_M, h')$ equal to (J, L), with $h' := \mathrm{Merkle}(h'_1, \ldots, h'_M)$,. The signature includes (J, L).

Step 3: For each $j \notin J$, the signer includes $\mathsf{mseed}^{[j]}$, h' in the signature. Also, for each $j \in J$, the signer includes $v^{[j]}$, $\tilde{x}^{[j]}$, $(\mathsf{seed}_i^{[j]})_{i \neq \ell_j}$, $c_{\ell_j}^{[j]}$ and $u_{\ell_j}^{[j]}$.

The only difference between this experiment and the previous one occurs if, in the course of answering a signing query, the query to Hash' in step 2 was ever made before (by either the adversary or as part of answering some other signing query). Letting $\mathrm{InputColl}_G$ denote this event, we have

$$|\Pr_3[\mathrm{Forge}] - \Pr_2[\mathrm{Forge}]| \leq \Pr_3[\mathrm{InputColl}_G].$$

Experiment 4. The difference with the previous experiment is that the signer now chooses uniform $\{\mathsf{seed}_i^{[j]}\}_{i \in [n]}$ for all $j \in J$. That is, signatures are now computed as follows:

Step 0:
- Choose uniform (J, L), where $J \subset [M]$ is a set of size τ, and $L = \{\ell_j\}_{j \in J}$ with $\ell_j \in [n]$.
- Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.
- Choose uniform $\mathsf{mseed}^{[0]} \in \{0,1\}^{\lambda}$.
- Compute the seeds $\mathsf{mseed}^{[1]}$, $\ldots$, $\mathsf{mseed}^{[M]}$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[0]})$.

Step 1: For each $j \in [M]$:

1. If $j \notin J$, use $\mathsf{mseed}^{[j]}$ to generate values $\mathsf{seed}_1^{[j]}$, ..., $\mathsf{seed}_n^{[j]}$ and $r^{[j]} \in \mathrm{Ker}(H)$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[j]})$. If $j \in J$, choose uniform $\mathsf{seed}_1^{[j]}$, ..., $\mathsf{seed}_n^{[j]} \in \{0,1\}^{\lambda}$ and $r^{[j]} \in \mathrm{Ker}(H)$.
2. For $i \in [n]$, sample $\sigma_i^{[j]}, s_i^{[j]}$ using $\mathsf{seed}_i^{[j]}$ and compute $c_i^{[j]} := \mathrm{Hash}_0(\mathsf{salt}, j, i, \mathsf{seed}_i^{[j]})$.
3. (Cut-and-choose phase) Compute

$$\begin{aligned} \sigma^{[j]} &:= \sigma_n^{[j]} \circ \cdots \circ \sigma_1^{[j]} \\ s^{[j]} &:= s_n^{[j]} + \sigma_n^{[j]}(\cdots + \sigma_2^{[j]}(s_1^{[j]})) \\ q^{[j]} &:= \sigma^{[j]}(r^{[j]}) + s^{[j]} \end{aligned}$$

4. (Sound phase) Compute

$$\begin{aligned} v^{[j]} &:= \sigma^{[j]}(x) \\ \tilde{x}^{[j]} &:= x + r^{[j]} \\ u_0^{[j]} &:= \tilde{x}^{[j]} \\ u_i^{[j]} &:= \sigma_i^{[j]}(u_{i-1}^{[j]}) + s_i^{[j]} \text{ for all } i \in [n] \end{aligned}$$

5. Compute $h_j := \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_n^{[j]})$ and $h'_j := \mathrm{Hash}_2(v^{[j]}, \tilde{x}^{[j]}, (u_i^{[j]})_i)$.

Step 2: Set $\mathrm{Hash}'(m, \mathsf{salt}, h_1, \ldots, h_M, h')$ equal to (J, L), with $h' := \mathrm{Merkle}(h'_1, \ldots, h'_M)$,. The signature includes (J, L).

Step 3: For each $j \notin J$, the signer includes $\mathsf{mseed}^{[j]}$, h' in the signature. Also, for each $j \in J$, the signer includes $v^{[j]}$, $\tilde{x}^{[j]}$, $(\mathsf{seed}_i^{[j]})_{i \neq \ell_j}$, $c_{\ell_j}^{[j]}$ and $u_{\ell_j}^{[j]}$.

It is easy to see that if the pseudorandom generator is $(t, \varepsilon_{\mathrm{PRG}})$-secure, then

$$|\Pr_4[\mathrm{Forge}] - \Pr_3[\mathrm{Forge}]| \leq q_s \cdot \tau \cdot \varepsilon_{\mathrm{PRG}}$$

and

$$|\Pr_4[\mathrm{InputColl}_G] - \Pr_3[\mathrm{InputColl}_G]| \leq q_s \cdot \tau \cdot \varepsilon_{\mathrm{PRG}}.$$

We now bound $\Pr_4[\text{InputColl}_G]$. Fix some previous query $(m, h_1, \ldots, h_M, h')$ to Hash', and look at a query $\text{Hash}'(\hat{m}, \hat{h}_1, \ldots, \hat{h}_M, \hat{h}')$ made while responding to some signing query. (In the rest of this discussion, we will use $\hat{\cdot}$ to represent values computed as part of answering that signing query.) For some fixed $j \in \hat{J}$, it is not hard to see that the probability of the event $\hat{h}_j = h_j$ is maximized if h_j was output by a previous query $\text{Hash}_1(q^{[j]}, c_1^{[j]}, ..., c_n^{[j]})$, and each $c_i^{[j]}$ was output by a previous $\text{Hash}_0(\mathsf{seed}_i^{[j]})$. (In all cases, the relevant prior query must be unique since the experiment is aborted if there is a collision in Hash_0 or Hash_1.) In that case, the probability that $\hat{h}_j = h_j$ is at most

$$(2^{-\lambda} + 2^{-2\lambda})^n + 2^{-2\lambda} \leq 2 \cdot 2^{-2\lambda}$$

(assuming $n \geq 3$), and thus the probability that $\hat{h}_j = h_j$ for all $j \in \hat{J}$ is at most $2^{-\tau \cdot (2\lambda - 1)}$. Taking a union bound over all signing queries and all queries made to Hash' (including those made during the course of answering signing queries), we conclude that

$$\Pr_4[\text{InputColl}_G] \leq q_s \cdot (q_s + q') \cdot 2^{-\tau \cdot (2\lambda - 1)}.$$

Experiment 5. The difference with the previous experiment is that:

- For each $j \in J$, choose uniform $c_{\ell_j}^{[j]}$ (*i.e.*, without making the corresponding query to Hash_0).
- For each $j \notin J$, choose uniform h_j' (*i.e.*, without making the corresponding query to Hash_2).

So, signatures are now computed as follows:

Step 0:
- Choose uniform (J, L), where $J \subset [M]$ is a set of size τ, and $L = \{\ell_j\}_{j \in J}$ with $\ell_j \in [n]$.
- Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.
- Choose uniform $\mathsf{mseed}^{[0]} \in \{0,1\}^{\lambda}$.
- Compute the seeds $\mathsf{mseed}^{[1]}, \ldots, \mathsf{mseed}^{[M]}$ with $\text{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[0]})$.

Step 1: For each $j \in [M]$:

1. If $j \notin J$, use $\mathsf{mseed}^{[j]}$ to generate values $\mathsf{seed}_1^{[j]}$, ..., $\mathsf{seed}_n^{[j]}$ and $r^{[j]} \in \text{Ker}(H)$ with $\text{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[j]})$. If $j \in J$, choose uniform $\mathsf{seed}_1^{[j]}$, ..., $\mathsf{seed}_n^{[j]} \in \{0,1\}^{\lambda}$ and $r^{[j]} \in \text{Ker}(H)$.
2. For $i \in [n]$, sample $\sigma_i^{[j]}, s_i^{[j]}$ using $\mathsf{seed}_i^{[j]}$ and compute

$$\begin{cases} c_{\ell_j}^{[j]} \text{ is chosen uniformly in } \{0,1\}^{2\lambda} \text{ if } j \in J \\ c_i^{[j]} := \text{Hash}_0(\mathsf{salt}, j, i, \mathsf{seed}_i^{[j]}) \text{ for all other } i, j \end{cases}.$$

3. (Cut-and-choose phase) Compute

$$\begin{aligned} \sigma^{[j]} &:= \sigma_n^{[j]} \circ \cdots \circ \sigma_1^{[j]} \\ s^{[j]} &:= s_n^{[j]} + \sigma_n^{[j]}(\cdots + \sigma_2^{[j]}(s_1^{[j]})) \\ q^{[j]} &:= \sigma^{[j]}(r^{[j]}) + s^{[j]} \end{aligned}$$

4. (Sound phase) Compute

$$\begin{aligned} v^{[j]} &:= \sigma^{[j]}(x) \\ \tilde{x}^{[j]} &:= x + r^{[j]} \\ u_0^{[j]} &:= \tilde{x}^{[j]} \\ u_i^{[j]} &:= \sigma_i^{[j]}(u_{i-1}^{[j]}) + s_i^{[j]} \text{ for all } i \in [n] \end{aligned}$$

5. Compute $h_j := \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_n^{[j]})$. For $j \in J$, set $h'_j := \mathrm{Hash}_2(v^{[j]}, \tilde{x}^{[j]}, (u_i^{[j]})_i)$; otherwise, choose uniform $h'_j \in \{0,1\}^{2\lambda}$.

Step 2: Set $\mathrm{Hash}'(m, \mathsf{salt}, h_1, \ldots, h_M, h')$ equal to (J, L), with $h' := \mathrm{Merkle}(h'_1, \ldots, h'_M)$,. The signature includes (J, L).

Step 3: For each $j \notin J$, the signer includes $\mathsf{mseed}^{[j]}$, h' in the signature. Also, for each $j \in J$, the signer includes $v^{[j]}$, $\tilde{x}^{[j]}$, $(\mathsf{seed}_i^{[j]})_{i \neq \ell_j}$, $c_{\ell_j}^{[j]}$ and $u_{\ell_j}^{[j]}$.

The only difference between this experiment and the previous one occurs if, during the course of answering a signing query, $\mathsf{seed}_{\ell_j}^{[j]}$ (for some $j \in J$) is queried to Hash_0 at some other point in the experiment, or $(v^{[j]}, \tilde{x}^{[j]}, (u_i^{[j]})_i)$ (for some $j \notin J$) is ever queried to Hash_2 at some other point in the experiment. Denoting this event by $\mathrm{InputColl}_H$, we thus have

$$|\Pr_5[\mathrm{Forge}] - \Pr_4[\mathrm{Forge}]| \leq \Pr_5[\mathrm{InputColl}_H].$$

Experiment 6. We again modify the experiment. Now, for $j \in J$ the signer uses the HVZK simulator (see Theorem 5), that we shall denote $\mathcal{S}$, to generate the views of the parties in an execution of a sound phase. This results in $\{\mathsf{seed}_i^{[j]}\}_{i \neq \ell_j}$, $q^{[j]}$, $v^{[j]}$, $\tilde{x}^{[j]}$ and $u_{\ell_j}^{[j]}$. From the respective views, $\{u_i^{[j]}\}_{i \neq \ell_j}$ can be computed, and h_j, h'_j can be computed as well. Thus, signatures are now computed as follows:

Step 0:
- Choose uniform (J, L), where $J \subset [M]$ is a set of size τ, and $L = \{\ell_j\}_{j \in J}$ with $\ell_j \in [n]$.
- Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.
- Choose uniform $\mathsf{mseed}^{[0]} \in \{0,1\}^{\lambda}$.
- Compute the seeds $\mathsf{mseed}^{[1]}, \ldots, \mathsf{mseed}^{[M]}$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[0]})$.

Step 1: For $j \notin J$:
1. Use $\mathsf{mseed}^{[j]}$ to generate values $\mathsf{seed}_1^{[j]}$, ..., $\mathsf{seed}_n^{[j]}$ and $r^{[j]} \in \mathrm{Ker}(H)$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[j]})$.
2. For $i \in [n]$, sample $\sigma_i^{[j]}, s_i^{[j]}$ using $\mathsf{seed}_i^{[j]}$ and compute $c_i^{[j]} := \mathrm{Hash}_0(\mathsf{salt}, j, i, \mathsf{seed}_i^{[j]})$.
3. (Cut-and-choose phase) Compute

$$\begin{aligned} \sigma^{[j]} &:= \sigma_n^{[j]} \circ \cdots \circ \sigma_1^{[j]} \\ s^{[j]} &:= s_n^{[j]} + \sigma_n^{[j]}(\cdots + \sigma_2^{[j]}(s_1^{[j]})) \\ q^{[j]} &:= \sigma^{[j]}(r^{[j]}) + s^{[j]} \end{aligned}$$

4. Let $h_j := \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, ..., c_n^{[j]})$. Choose uniform $h'_j \in \{0,1\}^{2\lambda}$.

For $j \in J$:
1. Compute $(\{\mathsf{seed}_i^{[j]}\}_{i \neq \ell_j}, q^{[j]}, v^{[j]}, \tilde{x}^{[j]}, u_{\ell_j}^{[j]}) \leftarrow \mathcal{S}(\ell_j)$. Compute $\{u_i^{[j]}\}_{i \neq \ell_j}$ based on this information.
2. Choose uniform $c_{\ell_j}^{[j]} \in \{0,1\}^{2\lambda}$. For all other i, set $c_i^{[j]} := \mathrm{Hash}_0(\mathsf{salt}, j, i, \mathsf{seed}_i^{[j]})$.
3. Let $h_j := \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, ..., c_n^{[j]})$ and $h'_j = \mathrm{Hash}_2(v^{[j]}, \tilde{x}^{[j]}, (u_i^{[j]})_i)$.

Step 2: Set $\mathrm{Hash}'(m, \mathsf{salt}, h_1, \ldots, h_M, h')$ equal to (J, L), with $h' := \mathrm{Merkle}(h'_1, \ldots, h'_M)$,. The signature includes (J, L).

Step 3: For each $j \notin J$, the signer includes $\mathsf{mseed}^{[j]}$, h' in the signature. Also, for each $j \in J$, the signer includes $v^{[j]}$, $\tilde{x}^{[j]}$, $(\mathsf{seed}_i^{[j]})_{i \neq \ell_j}$, $c_{\ell_j}^{[j]}$ and $u_{\ell_j}^{[j]}$.

Observe that the secret x is no longer used for generating signatures. Recall, the adversary against simulator has distinguishing advantage $\varepsilon_{\mathrm{PRG}} + \varepsilon_{\mathrm{Com}}$ where $\varepsilon_{\mathrm{Com}}$ is zero since we are in the Random Oracle Model and we remove the collisions in Experiment 2. It is immediate that

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

and

$$|\Pr_6[\text{InputColl}_H] - \Pr_5[\text{InputColl}_H]| \leq \tau \cdot q_s \cdot \varepsilon_{\text{PRG}}.$$

We now bound $\Pr_6[\text{InputColl}_H]$. For any particular signing query and any $j \in J$, the value $\mathsf{seed}_{\ell_j}^{[j]}$ has min-entropy at least λ and is not used anywhere else in the experiment. Similarly, for any $j \notin J$, the value $(v^{[j]}, \tilde{x}^{[j]})$ has min-entropy at least λ, since the input is λ-bit, and is not used anywhere else in the experiment. Thus,

$$\Pr_6[\text{InputColl}_H] \leq M \cdot q_s \cdot (Mq_s + q_0 + q_2) \cdot 2^{-\lambda}.$$

Experiment 7. We first define some notation. At any point during the experiment, we classify a pair (h_j, h'_j) in one of the following ways:

1. If h_j was output by a previous query $\text{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_n^{[j]})$, and each $c_i^{[j]}$ was output by a previous query $\text{Hash}_0(\mathsf{seed}_i^{[j]})$ where the $(\{\mathsf{seed}_i^{[j]}\}_i, q^{[j]})$ forms a valid preprocessing (*i.e.*, $\left(\sigma^{[j]}\right)^{-1}\left(q^{[j]} - s^{[j]}\right) \in \text{Ker}(H)$), then say (h_j, h'_j) *defines correct preprocessing.*
2. If h_j was output by a previous query $\text{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_n^{[j]})$, and each $c_i^{[j]}$ was output by a previous query $\text{Hash}_0(\mathsf{seed}_i^{[j]})$, and h'_j was output by a previous query $\text{Hash}_2(v^{[j]}, \tilde{x}^{[j]}, u_1^{[j]}, \ldots, u_n^{[j]})$ where $\{\mathsf{seed}_i^{[j]}\}_i$, $q^{[j]}$, $v^{[j]}$, $\tilde{x}^{[j]}$, $\{u_i^{[j]}\}_i$ are consistent with an online execution (but the $(\{\mathsf{seed}_i^{[j]}\}_i, q^{[j]})$ may not form a valid preprocessing), then say (h_j, h'_j) *defines correct execution.*
3. In any other case, say (h_j, h'_j) is *bad.*

(Note that in all cases the relevant prior query, if it exists, must be unique since the experiment is aborted if there is ever a collision in Hash_0, Hash_1, or Hash_2.)

In Experiment 7, for each query $\text{Hash}'(m, h_1, ..., h_M, \text{Merkle}(h'_1, \ldots, h'_M))$ made by the adversary (where m was not previously queried to the signing oracle), check if there exists an index j for which (h_j, h'_j) defines correct preprocessing and correct execution. We let Solve be the event that this occurs for some query to Hash'. Note that if that event occurs, the $\{\mathsf{seed}_i^{[j]}\}_i$, $q^{[j]}$, $v^{[j]}$, $\tilde{x}^{[j]}$ (which can be determined from the oracle queries of the adversary) allow computation of x' for which $Hx' = y$ and $\text{wt}(x') = w$. Thus, $\Pr_7[\mathsf{Solve}] \leq \varepsilon_{\text{SD}}$.

We claim that

$$\Pr_7[\text{Forge} \wedge \overline{\mathsf{Solve}}] \leq q' \cdot \varepsilon(M, n, \tau).$$

To see this, assume Solve does not occur. For any query $\text{Hash}'(m, h_1, ..., h_M,$
$\text{Merkle}(h'_1, \ldots, h'_M))$ made during the experiment (where m was not previously queried to the signing oracle), let Pre denote the set of indices for which (h_j, h'_j) defines correct preprocessing (but not correct execution), and let $k = |\mathsf{Pre}|$. Let (J, L) be the (random) answers from this query to Hash'. The attacker can only possibly generate a forgery (using this Hash'-query) if (1) $[M]\backslash J \subset \mathsf{Pre}$, and (2) for all $j \in \mathsf{Pre} \cap J$, the value ℓ_j is chosen to be the unique party such that the views of the remaining parties are consistent. Since $|M\backslash J| = M - \tau$, the number of ways the first event can occur is $\binom{k}{M-\tau}$; given this, there are $k - (M - \tau)$ elements remaining in $\mathsf{Pre} \cap J$. Thus, the overall probability with which the attacker can generate a forgery using this Hash'-query is

$$\begin{aligned}
\varepsilon(M, n, \tau, k) &= \frac{\binom{k}{M-\tau} \cdot n^{M-\tau}}{\binom{M}{M-\tau} \cdot n^{\tau}} \\
&= \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot n^{\tau - M + \tau}} \\
&\leq \varepsilon(M, n, \tau) := \max_k \{\varepsilon(M, n, \tau, k)\}.
\end{aligned}$$

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

□

G Boolean Circuit for Syndrome Decoding problem

Let us define a circuit for the boolean function $C_{H,y}$ defined as

$$C_{H,y} : x \in \mathbb{F}_2^m \mapsto \begin{cases} 1 \text{ if } Hx = y \text{ and } \mathrm{wt}(x) = w \\ 0 \text{ else} \end{cases} .$$

Since $H \in \mathbb{F}_2^{(m-k)\times m}$ is public, it is hardcoded into the circuit so that computing Hx is free in terms of AND gates. To compute $\mathrm{wt}(x)$, we need to sum all the bits of x, which involves AND gates to deal with carry propagation. To minimize the number of carries, and hence of AND gates, a possible strategy is to use a binary tree as follows:

- Let us denote $\ell := \lceil \log_2(w+1) \rceil$. For the sake of simplicity, we assume that m is a multiple of 2^ℓ.
- We split the m bits of x in t blocks of 2^ℓ bits.
- For each block of size 2^ℓ, we sum all the bits with a tree:
  - At the first level of the tree, we sum the bits two by two to obtain $\frac{2^\ell}{2}$ 2-bit values.
  - At the second level of the tree, we sum the previous 2-bit values two by two to obtain $\frac{2^\ell}{4}$ 3-bit values.

    $\vdots$

  - At the ℓ-th level of the tree, we sum the two previous $(\ell-1)$-bit values to obtain a single ℓ-bit value.

  The resulting number of AND gates for one 2^ℓ-bit block is

  $$2^{\ell-1} \cdot a(1) + 2^{\ell-2} \cdot a(2) + \ldots + 1 \cdot a(\ell)$$

  where $a(i)$ denotes the number of AND gates in an addition on i bits.
- We have t sum-blocks of ℓ bits. We now sum all these blocks using an additional overflow bit to keep in memory if the sum exceeds 2^ℓ. The resulting number of AND gates for this step is

  $$(t-1) \cdot a'(\ell)$$

  where $a'(\ell)$ denotes the number of AND gates in an addition on ℓ bits with the overflow bit.

So the total number of AND gates to compute $\mathrm{wt}(x)$ is

$$t \cdot \sum_{i=1}^{\ell} 2^{\ell-i} \cdot a(i) + (t-1) \cdot a'(\ell) .$$

For the addition circuit, we can use a full adder circuit and so

$$\begin{cases} a(i) = 1 + 2 \cdot (i-1) = 2i - 1 \\ a'(\ell) = 1 + 2 \cdot (\ell - 1) + 1 = 2\ell \end{cases} .$$

Once Hx and $\mathrm{wt}(x)$ are computed, the circuit just need to check that $Hx = y$ and $\mathrm{wt}(w)$, and for that it needs $(m-k) + l$ AND gates.