Designing a Practical Code-based Signature Scheme from Zero-Knowledge Proofs with Trusted Setup

Shay Gueron[1], Edoardo Persichetti[2], and Paolo Santini[3]

[1] University of Haifa, Israel
[2] Florida Atlantic University, USA
[3] Università Politecnica delle Marche, Italy
shay@math.haifa.ac.il, epersichetti@fau.edu, p.santini@staff.univpm.it

Abstract. This paper defines a new practical construction for a code-based signature scheme. We introduce a new protocol that is designed to follow the recent "Sigma protocol with helper" paradigm, and prove that the protocol's security reduces directly to the Syndrome Decoding Problem. The protocol is then converted to a full-fledged signature scheme via a sequence of generic steps that include: removing the role of the helper; incorporating a variety of protocol optimizations (using e.g., Merkle trees); applying the Fiat-Shamir transformation. The resulting signature scheme is EUF-CMA secure in the QROM, with the following advantages: a) Security relies on only minimal assumptions and is backed by a long-studied NP-complete problem; b) the trusted setup structure allows for obtaining an arbitrarily small soundness error. This minimizes the required number of repetitions, thus alleviating a major bottleneck associated with Fiat-Shamir schemes. We outline an initial performance estimation to confirm that our scheme greatly outpaces existing similar type solutions.

Keywords: Code-based, Signature, Zero-Knowledge, Syndrome Decoding

1 Introduction

Most of the public-key cryptosystems currently in use are threatened by the development of quantum computers. Due to Shor's algorithm [27], for example, the widely used RSA and ECC will be rendered insecure as soon as a large-scale functional quantum computer is built. To prepare for this scenario, there is a large body of active research aimed at providing alternative cryptosystems for which no quantum vulnerabilities are known. The earliest among those is the McEliece cryptosystem [23], which was introduced over four decades ago, and relies on the hardness of decoding random linear codes. Indeed, a modern rendition of McEliece's scheme [4] is one of the major players in NIST's recent standardization effort for quantum-resistant public-key cryptographic schemes [1].

Lattice-based cryptosystems play a major role in NIST's process, especially due to their impressive performance figures. Yet, code-based cryptography, the area comprising the McEliece cryptosystem and its offspring, provides credible candidates for the task of key establishment (other examples being HQC [24] and BIKE [5], both admitted to the final round as "alternates"). The situation, however, is not the same when talking about digital signatures. Indeed, NIST identified a shortage of alternatives to lattice-based candidates, to the point of planning a reopening of the call for proposals (see for instance [2]).

There is a long history of code-based signatures, dating back to Stern's work [28], which introduced a zero-knowledge identification scheme (ZKID). It is known that ZKIDs can be turned into full-fledged signature schemes via the Fiat-Shamir transformation [17]. Stern's first proposal has since been extended, improved and generalized (e.g., [29, 18, 13]). However, all of these proposals share a common drawback: a high soundness error, ranging from 2/3 to (essentially) 1/2. This implies that the protocol requires multiple repetitions and leads to very long signatures. Other schemes, based on different techniques, have also been proposed in literature. These include trapdoor-based constructions (e.g., [14, 15]) and more recently, protocols based on code equivalence (e.g., [12, 7]). All of the schemes have different features, strengths and drawbacks, and it is fair to say that none of them is overall fully satisfactory.

Our Contribution. We present a new code-based zero-knowledge scheme that substantially improves on the existing literature by featuring an arbitrarily low soundness error. This allows to greatly reduce the number of repetition rounds needed to obtain the target security level, consequently reducing the signature size. We equip our scheme with a wide range of optimizations, leading to improved performance. Our construction leverages the recent approach by Katz et al. [20], using a Multi Party Computation (MPC) protocol with preprocessing. More concretely, we design a "Sigma protocol with helper", following the nomenclature of [10]. We then show how to convert it to a full-fledged signature scheme and provide an in-depth analysis of the various techniques utilized to refine the protocol.

2 Preliminaries

2.1 Coding Theory

An $[n, k]$-*linear code* $\mathfrak{C}$ of length n and dimension k over $\mathbb{F}_q$ is a k-dimensional vector subspace of $\mathbb{F}_q^n$. It is usually represented in one of two ways. The first identifies a matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$, called *generator matrix*, whose rows form a basis for the vector space, i.e., $\mathfrak{C} = \{\mathbf{uG},\ \ \mathbf{u} \in \mathbb{F}_q^k\}$. The second way, instead, describes the code as the kernel of a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$, called *parity-check matrix*, i.e. $\mathfrak{C} = \{\mathbf{x} \in \mathbb{F}_q^n : \mathbf{xH}^\top = 0\}$. Linear codes are usually measured using the Hamming weight, which corresponds to the number of non-zero positions in a vector. Isometries for the Hamming metric are given by monomial transformations $\tau = (\mathbf{v}; \pi) \in \mathbb{F}_q^{*n} \rtimes \mathsf{S}_n$, i.e. permutation combined with scaling factors. In this paper, we will denote by $\mathsf{FWV}(q, n, w)$ the set of vectors of length n with elements in $\mathbb{F}_q$ and fixed Hamming weight w. The parity-check representation for a linear code leads to the following well-known problem.

Definition 1 (Syndrome Decoding Problem (SDP)). *Let $\mathfrak{C}$ be a code of length n and dimension k defined by a parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$, and let $\mathbf{s} \in \mathbb{F}_q^{n-k}$, $w \leq n$. Find $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{eH}^\top = \mathbf{s}$ and $\mathbf{e}$ is of weight w.*

SDP was proved to be NP-complete for both the binary and q-ary cases [9, 8], and is thus a solid base to build cryptography on. In fact, the near-entirety of code-based cryptography relies more or less directly on SDP. The main solvers for SDP belong to the family of Information-Set Decoding (ISD); we will expand on this when discussing practical instantiations and parameter choices, in Section 5.

2.2 Technical Tools

We recall here the characteristics of a Sigma protocol with helper, as formalized in [10]. This is an interactive protocol between three parties: a *prover*, a *verifier* and a trusted third party called *helper*. The prover holds a secret sk corresponding to a public key pk, which is held by the verifier. The protocol takes place in the following phases:

I. Precomputation: the helper takes a random seed seed as input, generates some auxiliary information aux, then sends the former to the prover and the latter to the verifier.

II. Commitment: the prover uses seed, in addition to his secret sk, to create a commitment c and sends it to the verifier.

III. Challenge: the verifier selects a random challenge ch from the challenge space and sends it to the prover.

IV. Response: the prover computes a response rsp using ch (in addition to his previous information), and sends it to the verifier.

V. Verification: the verifier checks the correctness of rsp, then checks that this was correctly formed using aux, and accepts or rejects accordingly.

A Sigma protocol with helper is expected to satisfy the following properties:

– **Completeness:** if all parties follow the protocol correctly, then the verifier always accepts.

– **2-Special Soundness:** given an adversary $\mathcal{A}$ that outputs two valid transcripts $(\mathsf{aux}, \mathsf{c}, \mathsf{ch}, \mathsf{rsp})$ and $(\mathsf{aux}, \mathsf{c}, \mathsf{ch}', \mathsf{rsp}')$ with $\mathsf{ch} \neq \mathsf{ch}'$, it is possible to extract a valid secret[4] sk'.

– **Special Honest-Verifier Zero-Knowledge:** there exists a probabilistic polynomial-time simulator algorithm that is capable, on input $(\mathsf{pk}, \mathsf{seed}, \mathsf{ch})$, to output a transcript $(\mathsf{aux}, \mathsf{c}, \mathsf{ch}, \mathsf{rsp})$ which is computationally indistinguishable from one obtained via an honest execution of the protocol.

Of course, the existence of a helper party fitting the above description is not realistic, and thus it is to be considered just a technical tool to enable the design of the protocol. In Section 3.1, we will show the details of the transformation to convert a sigma protocol with helper into a customary 3-pass ZK protocol.

The design of such a protocol will crucially rely on several building blocks, in addition to those coming from coding theory. Besides common ones such as hash functions, as well as pseudorandom functions and number generators, we utilize two additional types of objects. The first, is a *keyed* hash function $\mathsf{K} : \{0,1\}^\lambda \times \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$, following the approach of hash-based signatures such as XMSS-T and SPHINCS+ [19, 6]. With this tool, when building a Merkle tree, the security will depend on the second-preimage resistance of this function, rather than on the collision resistance of a "regular" hash function. This allows to avoid collision attacks, and, consequently, to utilize hash digests that are of length λ, rather than 2λ. Secondly, we employ a non-interactive commitment function $\mathsf{Com} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$. The first λ bits of input, chosen uniformly at random, guarantee that the input message is hidden in a very strong sense, as captured in the next definition.

Definition 2. *Given an adversary $\mathcal{A}$, we define the two following quantities:*

$$\mathsf{Adv}^{\mathsf{Bind}}(\mathcal{A}) = \mathsf{Pr}\Big[\mathsf{Com}(\mathsf{r}, \mathbf{x}) = \mathsf{Com}(\mathsf{r}', \mathbf{x}') \mid (\mathsf{r}, \mathbf{x}, \mathsf{r}', \mathbf{x}') \leftarrow \mathcal{A}(1^\lambda)\Big];$$

$$\mathsf{Adv}^{\mathsf{Hide}}(\mathcal{A}, \mathbf{x}, \mathbf{x}') = \Big|\underset{\mathsf{r} \leftarrow \{0,1\}^\lambda}{\mathsf{Pr}}\Big[\mathcal{A}(\mathsf{Com}(\mathsf{r}, \mathbf{x})) = 1\Big] - \underset{\mathsf{r} \leftarrow \{0,1\}^\lambda}{\mathsf{Pr}}\Big[\mathcal{A}(\mathsf{Com}(\mathsf{r}, \mathbf{x}')) = 1\Big]\Big|.$$

We say that Com *is* computationally binding *if, for all polynomial-time adversaries $\mathcal{A}$, the quantity $\mathsf{Adv}^{\mathsf{Bind}}(\mathcal{A})$ is negligible in λ. We say that* Com *is* computationally hiding *if, for all polynomial-time adversaries $\mathcal{A}$ and every pair $(\mathbf{x}, \mathbf{x}')$, the quantity $\mathsf{Adv}^{\mathsf{Hide}}(\mathcal{A})$ is negligible in λ.*

Informally, the two properties defined above ensure that nothing about the input is revealed by the commitment and that, furthermore, it is infeasible to open the commitment to a different input.

[4] This is not necessarily the one held by the prover, but could in principle be any witness for the relation $(\mathsf{pk}, \mathsf{sk})$.

3 The New Scheme

Below, we give a schematic description of the new Sigma protocol with helper.

Public Data
Parameters $q, n, k, w \in \mathbb{N}$, a full-rank matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ and a commitment function $\mathsf{Com} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$.

Private Key
A vector $\mathbf{e} \in \mathsf{FWV}(q, n, w)$.

Public Key
The syndrome $\mathbf{s} = \mathbf{e}\mathbf{H}^\top$.

I. Precomputation (Helper)
Input: -

1. Sample uniform random $\mathsf{seed} \in \{0,1\}^\lambda$.
2. Generate $\mathbf{u} \in \mathbb{F}_q^n$ and $\tilde{\mathbf{e}} \in \mathsf{FWV}(q, n, w)$ from seed.
3. For all $v \in \mathbb{F}_q$:
   i. Generate randomness $\mathsf{r}_v \in \{0,1\}^\lambda$ from seed.
   ii. Compute $\mathsf{c}_v = \mathsf{Com}(\mathsf{r}_v, \mathbf{u} + v\tilde{\mathbf{e}})$.
4. Set $\mathsf{aux} = \{\mathsf{c}_v\}$.
5. Send seed to prover and aux to verifier.

II. Commitment (Prover)
Input: $\mathbf{H}, \mathbf{e}$ and seed.

1. Regenerate $\mathbf{u} \in \mathbb{F}_q^n$ and $\tilde{\mathbf{e}} \in \mathsf{FWV}(q, n, w)$ from seed.
2. Determine isometry τ such that $\mathbf{e} = \tau(\tilde{\mathbf{e}})$.
3. Generate randomness $\mathsf{r} \in \{0,1\}^\lambda$.
4. Compute $\mathsf{c} = \mathsf{Com}(\mathsf{r}, \tau, \tau(\mathbf{u})\mathbf{H}^\top)$.
5. Send c to verifier.

III. Challenge (Verifier)
Input: -

1. Sample uniform random $z \in \mathbb{F}_q$.
2. Send z to prover.

IV. Response (Prover)
Input: z and seed.

1. Regenerate r_z from seed.
2. Compute $\mathbf{y} = \mathbf{u} + z\tilde{\mathbf{e}}$.
3. Send $\mathsf{rsp} = (\mathsf{r}, \mathsf{r}_z, \tau, \mathbf{y})$ to verifier.

V. Verification (Verifier)
Input: $\mathbf{H}, \mathbf{s}, \mathsf{aux}, \mathsf{c}$ and rsp.

1. Compute $\mathbf{t} = \tau(\mathbf{y})\mathbf{H}^\top - z\mathbf{s}$.
2. Check that $\mathsf{Com}(\mathsf{r}, \tau, \mathbf{t}) = \mathsf{c}$ and that τ is an isometry.
3. Check that $\mathsf{Com}(\mathsf{r}_z, \mathbf{y}) = \mathsf{c}_z$.
4. Accept if both checks are successful, or reject otherwise.

We now proceed to prove that the above identification schemes satisfies the three fundamental properties of a Sigma protocol with helper, which we described in Section 2.2.

Correctness: we have that $\tau(\mathbf{y})\mathbf{H}^\top = \tau(\mathbf{u} + z\tilde{\mathbf{e}})\mathbf{H}^\top = \tau(\mathbf{u})\mathbf{H}^\top + z\tau(\tilde{\mathbf{e}})\mathbf{H}^\top$ and the second addendum is exactly $z\mathbf{s}$, which yields $\tau(\mathbf{u})\mathbf{H}^\top$ as expected.

Intuitively, security is based on the fact that enforcing τ to be an isometry is equivalent to checking that $\tilde{\mathbf{e}}$ has the correct weight: this is captured in the next proof.

Soundness: we want to show that, given two transcripts that differ in the challenge, we are able to extract a solution for SDP. Consider then the two transcripts $(\mathsf{aux}, \mathsf{c}, z, \mathsf{r}, \mathsf{r}_z, \tau, \mathbf{y})$ and $(\mathsf{aux}, \mathsf{c}, z', \mathsf{r}', \mathsf{r}_{z'}, \tau', \mathbf{y}')$ with $z \neq z'$. Now let $\mathbf{t} = \tau(\mathbf{y})\mathbf{H}^\top - z\mathbf{s}$ and $\mathbf{t}' = \tau'(\mathbf{y}')\mathbf{H}^\top - z'\mathbf{s}$. By the binding properties of the commitment (hash etc.), the verifier only accepts if $\mathsf{c} = \mathsf{Com}(\mathsf{r}, \tau, \mathbf{t}) = \mathsf{Com}(\mathsf{r}', \tau', \mathbf{t}')$, so in particular this implies $\tau = \tau'$ and $\mathbf{t} = \mathbf{t}'$. Since the helper computed everything honestly, and aux is properly formed, the verifier also requires that $\mathsf{c}_z = \mathsf{Com}(\mathsf{r}_z, \mathbf{u} + z\tilde{\mathbf{e}}) = \mathsf{Com}(\mathsf{r}_z, \mathbf{y})$ and $\mathsf{c}_{z'} = \mathsf{Com}(\mathsf{r}_{z'}, \mathbf{u} + z'\tilde{\mathbf{e}}) = \mathsf{Com}(\mathsf{r}_{z'}, \mathbf{y}')$, from which it follows, respectively, that $\mathbf{y} = \mathbf{u} + z\tilde{\mathbf{e}}$ and $\mathbf{y}' = \mathbf{u} + z'\tilde{\mathbf{e}}$. We now put all the pieces together, starting from $\mathbf{t} = \mathbf{t}'$, $\tau = \tau'$ and substituting in. We obtain that $\tau(\mathbf{u} + z\tilde{\mathbf{e}})\mathbf{H}^\top - z\mathbf{s} = \tau(\mathbf{u} + z'\tilde{\mathbf{e}})\mathbf{H}^\top - z'\mathbf{s}$, which implies that $z\tau(\tilde{\mathbf{e}})\mathbf{H}^\top - z\mathbf{s} = z'\tau(\tilde{\mathbf{e}})\mathbf{H}^\top - z'\mathbf{s}$. Rearranging and gathering common terms leads to $(z - z')\tau(\tilde{\mathbf{e}})\mathbf{H}^\top = (z - z')\mathbf{s}$ and since $z \neq z'$, we conclude that $\tau(\tilde{\mathbf{e}})\mathbf{H}^\top = \mathbf{s}$. It follows that $\tau(\tilde{\mathbf{e}})$ is a solution to SDP as desired. Note that this can easily be calculated since $\mathbf{y} - \mathbf{y}' = (z - z')\tilde{\mathbf{e}}$, from which $\tilde{\mathbf{e}}$ can be obtained and hence $\tau(\tilde{\mathbf{e}})$ (since τ is known).

Zero-knowledge: it is easy to prove that there is no knowledge leaked by honest executions. To show this, we construct a simulator which proceeds as follows. Take as input $\mathbf{H}, \mathbf{s}$ a challenge z and the seed. The simulator then reconstructs aux, r_z and $\mathbf{y} = \mathbf{u} + z\tilde{\mathbf{e}}$, having obtained $\mathbf{u}$ and $\tilde{\mathbf{e}}$ from the seed. It then selects a random permutation π and computes $\mathbf{t} = \pi(\mathbf{y})\mathbf{H}^\top - z\mathbf{s}$. Finally, it generated a randomness r, calculates the commitment as $\mathsf{c} = \mathsf{Com}(\mathsf{r}, \pi, \mathbf{t})$, and outputs a transcript $(\mathsf{aux}, \mathsf{c}, z, \mathsf{r}, \mathsf{r}_z, \pi, \mathbf{y})$. It is easy to see that such a transcript passes verification, and in fact it is identical to the one produced by an honest prover, with the exception of π being used instead of τ.

3.1 Removing the Helper

We now explain how the protocol above can be converted into a standard Zero-Knowledge protocol (without the artificial helper). The main idea is to use a "cut-and-choose" technique, as suggested by Katz et al. [20]. The simplest way to accomplish this is the following. The prover can simulate the work of the helper by performing all the duties of the precomputation phase; namely, the prover is able to generate a seed, run the setup process to obtain aux and generate the protocol commitment c. The verifier can then hold the prover accountable by asking to do this several times, and then querying on a single random instance,

that gets executed. The other instances still get checked, by simply using the respective seeds, which get transmitted along with the protocol response of the one selected instance. To be more precise, we give below a schematic description of the new scheme, which we call $\mathcal{P}'$; to keep the notation as simple as possible, we use a generic notation and summarize several steps, referring to the previous scheme, called $\mathcal{P}$, as needed.

Public Data, Private Key, Public Key

Same as in $\mathcal{P}$.

I. Commitment (Prover)

Input: Public data and private key.

1. For all $i = 0, \ldots, N-1$:
   i. Sample uniform random $\mathsf{seed}^{(i)} \in \{0,1\}^\lambda$.
   ii. Perform precomputation steps I.2. – I.4. from $\mathcal{P}$ to get $\mathsf{aux}^{(i)}$.
   iii. Perform commitment steps II.2. – II.4. from $\mathcal{P}$ to get $\mathsf{c}^{(i)}$.
2. Send $\mathsf{aux}^{(0)}, \ldots, \mathsf{aux}^{(N-1)}$ and $\mathsf{c}^{(0)}, \ldots, \mathsf{c}^{(N-1)}$ to verifier.

II. Challenge (Verifier)

Input: -

1. Sample uniform random index $I \in \{0, \ldots, N-1\}$.
2. Sample uniform random challenge ch.
3. Send I and z to prover.

III. Response (Prover)

Input: I, ch and $\mathsf{seed}^{(I)}$.

1. Use ch and $\mathsf{seed}^{(I)}$ to compute response $\mathsf{rsp}^{(I)}$ for instance I.
2. Send rsp and $\{\mathsf{seed}^{(i)}\}_{i \neq I}$ to verifier.

IV. Verification (Verifier)

Input: $\mathbf{H}, \mathbf{s}, \mathsf{aux}^{(0)}, \ldots, \mathsf{aux}^{(N-1)}, \mathsf{c}^{(0)}, \ldots, \mathsf{c}^{(N-1)}, \mathsf{rsp}^{(I)}$ and $\{\mathsf{seed}^{(i)}\}_{i \neq I}$.

1. For all $i \neq I$:
   i. Use $\mathsf{seed}^{(i)}$ to recompute $\mathsf{aux}^{(i)}$.
   ii. Check that this matches the value sent in step I.2.
2. Perform verification steps V.2. – V.3. from $\mathcal{P}$, for instance I.
3. Accept if all checks in steps 1.ii. and 2. are successful, or reject otherwise.

It is possible to see [11, Theorem 3] that the modified protocol yields a Zero-Knowledge proof of knowledge with soundness error $\varepsilon = \max\{1/N; 1/C\}$, where C is the cardinality of the challenge space: in our case the challenge is a value $z \in \mathbb{F}_q$, and therefore we have $C = q$. The protocol can then be iterated, as customary, to obtain the desired soundness error of $2^{-\lambda}$; namely, the protocol is repeated t times, with $t = \lceil -\lambda / \log \varepsilon \rceil$. Katz et al. [20] also show how it is possible to beat parallel repetition, by using a more sophisticated approach. In order to have a clearer description of the costs, below, we postpone the discussion on this approach until the end of the next section.

4 Communication Cost and Optimizations

Several optimizations are presented in [11], albeit in a rather informal way. Moreover, these are all combined together and nested into each other, so that, in the end, it is quite hard to have a clear view of the full picture. In here, we strive to present the optimizations in full detail, one at a time, show how they can all be applied to our protocol, and illustrate their impact on the communication cost. To begin, we analyze the communication cost of a single iteration of the protocol, before any optimizations are applied. This consists of several components:

- N copies of the auxiliary information $\mathsf{aux}^{(i)}$, each consisting of q hash values;
- N copies of the commitment $\mathsf{c}^{(i)}$, each consisting of a single hash value;
- the index I and the challenge z, respectively an integer and a field element;
- the protocol response $(\mathsf{r}, \mathsf{r}_z, \tau, \mathbf{y})$, consisting of two λ-bit randomness strings, an isometry, and a length-n vector with elements in $\mathbb{F}_q$;
- the $N-1$ seeds $\{\mathsf{seed}^{(i)}\}_{i \neq I}$, each a λ-bit string.

This leads to the following formula for the communication cost (in bits)

$$\underbrace{2\lambda q N}_{\{\mathsf{aux}^{(i)}\}} + \underbrace{2\lambda N}_{\{\mathsf{c}^{(i)}\}} + \underbrace{\lceil \log N \rceil}_{I} + \underbrace{\lceil \log q \rceil}_{z} + \underbrace{2\lambda}_{\mathsf{r},\mathsf{r}_z} + \underbrace{n\big(\lceil \log n \rceil + \lceil \log q \rceil\big)}_{\tau} + \underbrace{n\lceil \log q \rceil}_{\mathbf{y}} + \underbrace{\lambda(N-1)}_{\{\mathsf{seed}^{(i)}\}_{i \neq I}}, \tag{1}$$

where all logarithms are assumed to be in base 2. Note that we have chosen to leave the above formula in its unsimplified version, in order to highlight all the various components. This will be helpful when analyzing the impact of the different optimizations.

Protocol Commitments. The first optimization that we discuss regards the commitments $\mathsf{c}^{(i)}$; we choose to present this first, because it involves the first verifier check, and also because it can serve as a blueprint for other optimizations. Now, note that the prover transmits N copies of $\mathsf{c}^{(i)}$, but only one of them is actually employed in the verification (the one corresponding to instance I). It follows that the transmission cost can be reduced, by employing a Merkle tree T of depth $d = \lceil \log N \rceil$, whose leaves are associated to $\mathsf{c}^{(0)}, \ldots, \mathsf{c}^{(N-1)}$.

To be more precise, as we anticipated, we will employ the *keyed* hash function $\mathsf{K} : \{0,1\}^\lambda \times \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ that we introduced in Section 2.2. The tree is created by setting

$$T_{u,l} = \mathsf{K}(\mathsf{PRF}(\mathsf{seed}, 2^u + l), (T_{u+1,2l} || T_{u+1,2l+1}) \oplus \mathsf{PRF}'(\mathsf{seed}, 2^u + l))$$

for $0 \leq u \leq d$ and $0 \leq l \leq 2^u - 1$, where $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^{d+1} \to \{0,1\}^\lambda$ and $\mathsf{PRF}' : \{0,1\}^\lambda \times \{0,1\}^{d+1} \to \{0,1\}^{2\lambda}$ are two pseudo-random functions, starting from the leaves that are defined as

$$T_{d,l} = \mathsf{K}(\mathsf{PRF}(\mathsf{seed}, 2^d + l), \mathsf{c}_v \oplus \mathsf{PRF}'(\mathsf{seed}, 2^d + l))$$

for $0 \leq l \leq 2^d - 1$. Only seed and the root of the tree need to be initially transmitted to the verifier; the prover will then include the authentication path of the tree in his response, after receiving the challenge. By *authentication path*, we mean the list of the hash values corresponding to the siblings of the nodes on the path from c_j to the root. This means that the component $2\lambda N$ in Equation (1) is reduced to $2\lambda + \lambda\lceil \log N \rceil$.

Auxiliary Information. We now illustrate how to deal with the cost of transmitting the N copies of the auxiliary information $\mathsf{aux}^{(i)}$. This can be greatly reduced, using two distinct optimizations.

First, notice that only one out of the q commitment values is employed in a single protocol execution, namely, in the second verifier check. This means that we can again use a Merkle tree, with a setup similar as the previous optimization; in this case, the leaves will be associated to the values $\{\mathsf{c}_v\}$. Thus, once more, only the root (and corresponding seed) need to be transmitted initially, and the authentication path can be included in the response phase. Accordingly, the component $2\lambda qN$ in Equation (1) is reduced to $N(2\lambda + \lambda\lceil \log q \rceil)$.

Furthermore, we can look at the previous improvement in another light: only one of the N instances is actually executed, while the other ones are simply checked by recomputing the setups using the seeds. Thus, there is no need to transmit all N copies of $\mathsf{aux}^{(i)}$, even in the above simplified version consisting of root + path. Instead, the prover can compute a hash of the roots, send it to the verifier, and include in his response only the authentication path for instance I. In the verification phase, the root for instance I is computed via protocol execution, while the other roots are recomputed via the seeds $\{\mathsf{seed}^{(i)}\}_{i \neq I}$; then, the verifier hashes the newly-computed roots and checks the resulting value against the transmitted one. In the end, with this second technique, the component $N(2\lambda + \lambda\lceil \log q \rceil)$ that we previously obtained is reduced to $2\lambda + \lambda\lceil \log q \rceil$.

Seeds. We are going to use a binary tree again, one last time, to reduce the communication cost associated with the $N - 1$ seeds sent by the prover along with his response. However, in this case, the tree is built starting from the root (i.e., the opposite of what one does to build Merkle trees). The prover begins by choosing a random seed to be the root of the tree. He then uses a pseudo-random generator $\mathsf{PRNG} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ that, on input an internal node $T_{u,l}$ returns $(T_{u+1,2l} || T_{u+1,2l+1}) = \mathsf{PRNG}(T_{u,l})$, for $0 \leq u \leq d$ and $0 \leq l \leq 2^u - 1$, where $d = \log N$. In the end, the tree will produce N values as leaves, which are going to be exactly the N seeds used in the protocol. Now, one seed is used and the remaining $N - 1$ need to be made available to the verifier. Indeed, seed_I should *not* be revealed, so the prover cannot transmit the root of the tree. Instead, the prover transmits the d internal nodes that are "companions", i.e. siblings to the parents (and grandparents etc.) of seed_I. With this technique, the component $\lambda(N - 1)$ in Equation (1) is reduced to just $\lambda\lceil \log N \rceil$.

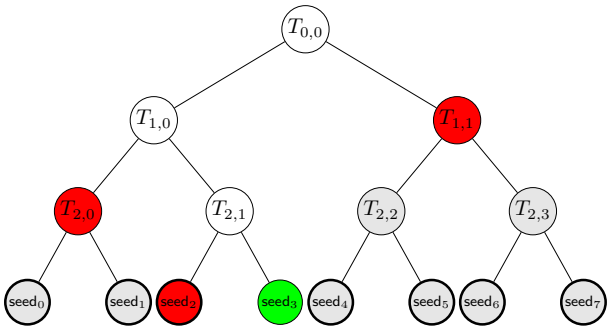

Fig. 1: Example of binary tree for $N = 8$. The chosen seed (in green) is used and not revealed. The prover transmits the red nodes and the verifier can generate the remaining seeds (but not the chosen one) by applying PRNG to $T_{2,0}$ and $T_{1,1}$ (and hence to $T_{2,2}$ and $T_{2,3}$). The nodes generated in this way are colored with in gray. The leaves obtained are highlighted with the thick line.

Executions. We are now ready to discuss the more sophisticated approach of Katz et al. [20], which will yield better performance compared to a simple parallel repetion of the protocol. The main idea is to modify the "cut-and-choose" technique as follows. Currently, the protocol precomputes N distinct instances and only executes one, then repeats this process t times; thus, one has to precompute a total of tN instances, out of which only t are executed. As mentioned above, this leads to a soundness error of $2^{-\lambda} = \varepsilon^t$, where $\varepsilon = \max\{1/N, 1/q\}$. $\varepsilon = 1/N$. Instead, the same soundness error can be obtained by having a larger number of instances, say M, but executing a subset of them, rather than just one; this entirely avoids the need for repetition. In fact, as explained in [11], the soundness error, in this case, is bounded above by

$$\max_{e \in [0,s]} \frac{\binom{M-e}{s-e}}{\binom{M}{s} q^{s-e}} \tag{2}$$

where s is the number of executed instances (which are indexed by a set $S \subseteq \{0, \ldots, M-1\}$) and $e \leq s$ is a parameter that indicates how many instances are incorrectly precomputed by an adversary. Note that, for these instances, the adversary would not be able to produce values seed_i, and therefore, the only chance he has to win, is if the verifier chooses to execute exactly all such instances; this happens with probability equal to $\binom{M-e}{s-e}/\binom{M}{s}$. The remaining term in Equation (2) is given by the probability of answering correctly (which is $1/q$) in each of the remaining $s - e$ instances. For a formal proof of this fact, we refer the reader to [20].

In terms of communication cost, the total amount for the method using plain parallel repetition is given by t times the cost of one execution, refined with the

previous optimizations. This is given (in bits) by

$$t\Big(\underbrace{2\lambda+\lambda\lceil\log q\rceil}_{\{\mathsf{aux}^{(i)}\}}+\underbrace{2\lambda+\lambda\lceil\log N\rceil}_{\{\mathsf{c}^{(i)}\}}+\underbrace{\lambda\lceil\log N\rceil}_{\{\mathsf{seed}^{(i)}\}_{i\neq I}}+\mathsf{C}_{cr}\Big) \tag{3}$$

where we have simplified to $\mathsf{C}_{cr}=2\lambda+\lceil\log N\rceil+n\lceil\log n\rceil+(2n+1)\lceil\log q\rceil$ the cost of transmitting the challenge and response, which is fixed. In comparison, with the new method, the various tree roots need only be transmitted once. This would lead to the following total cost (again, in bits)

$$\underbrace{2\lambda+s\lambda\lceil\log q\rceil}_{\{\mathsf{aux}^{(i)}\}}+\underbrace{2\lambda+s\lambda\lceil\log M\rceil}_{\{\mathsf{c}^{(i)}\}}+\underbrace{s\lambda\lceil\log M\rceil}_{\{\mathsf{seed}^{(i)}\}_{i\notin S}}+s\mathsf{C}'_{cr} \tag{4}$$

with $\mathsf{C}'_{cr}=2\lambda+\lceil\log M\rceil+n\lceil\log n\rceil+(2n+1)\lceil\log q\rceil$. While the number of instances necessarily increases (i.e. $M>N$), the number of executions remains about the same as for the case of parallel repetition (i.e. $s\approx t$). Since the former number appears only logarithmically in the cost, while the latter is linear, the above optimization allows to greatly reduce the total number of setups needed (from tN down to M) without increasing communication cost.

It is worth commenting on the behavior of some of the logarithmic terms in Equation (4), which correspond to the different trees used in the various optimizations. First of all, since the executed instances are chosen among the same set, all of the commitments $\{\mathsf{c}^{(i)}\}$ are taken from a single tree. In this case, the different authentication paths will present some common nodes, and one could think that fewer nodes are necessary in practice. However, considering the ratio of s to M, such an intersection is rather small and would probably not lead to a meaningful reduction in cost. Thus, for ease of analysis, we choose to leave the term $s\lambda\lceil\log M\rceil$ as it is, which is an upper bound.

For the tree of the $\{\mathsf{seed}^{(i)}\}$, instead, there is a noticeable cost reduction. In this case, in fact, it is not necessary to send multiple paths. Instead, the required $M-s$ seeds can all be obtained in batch using a variable number of nodes, which depends on the position of the chosen leaves; an illustration is given in Figure 2. With simple computations, one can find that, in the worst case, the number of nodes which must be transmitted is given by

$$2^{\lceil\log(s)\rceil}+s(\lceil\log(M)\rceil-\lceil\log(s)\rceil-1).$$

Conservatively, we will then estimate the cost as λ times this number, and substitute this in Equation (4), obtaining the final cost formula (again, in bits):

$$\underbrace{2\lambda+s\lambda\lceil\log q\rceil}_{\{\mathsf{aux}^{(i)}\}}+\underbrace{2\lambda+s\lambda\lceil\log M\rceil)}_{\{\mathsf{c}^{(i)}\}} + \underbrace{\lambda\big(2^{\lceil\log(s)\rceil}+s(\lceil\log(M)\rceil-\lceil\log(s)\rceil-1)\big)}_{\{\mathsf{seed}^{(i)}\}_{i\notin S}}+s\mathsf{C}'_{cr}. \tag{5}$$

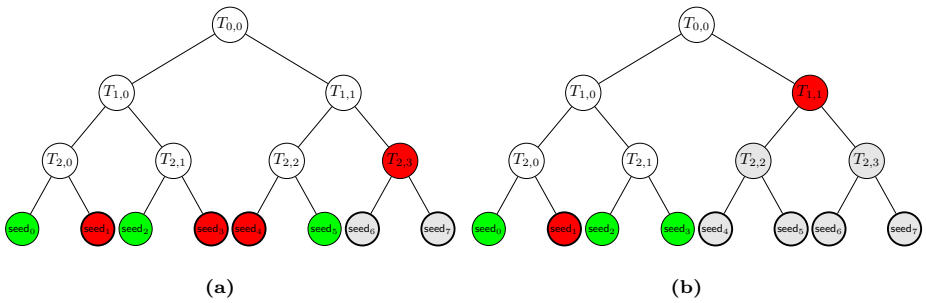

Fig. 2: Example of two binary trees for $M = 8, s = 3$. Color codes are the same as in Figure 1. For tree (a), it is necessary to transmit 4 nodes, whereas for tree (b), only 2 nodes need to be transmitted.

To give a complete picture, that sums up all the optimizations we consider, we present a schematic description below.

Public Data

Parameters $q, n, k, w \in \mathbb{N}$, a full-rank matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, a commitment function $\mathsf{Com} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{2\lambda}$ and a hash function $\mathsf{Hash} : \{0,1\}^{\lambda M} \to \{0,1\}^{2\lambda}$.

Private Key

A vector $\mathbf{e} \in \mathsf{FWV}(q, n, w)$.

Public Key

The syndrome $\mathbf{s} = \mathbf{e}\mathbf{H}^\top$.

I. Commitment (Prover)

Input: $\mathbf{H}$ and $\mathbf{e}$.

1. Generate tree with $\mathsf{seed}^{(0)}, \ldots, \mathsf{seed}^{(M-1)}$ as leaves.
2. For all $i = 0, \ldots, M-1$:
   i. Perform precomputation steps I.2. – I.4. from $\mathcal{P}$ to get $\mathsf{aux}^{(i)}$.
   ii. Build tree $T^{(i)}_{\mathsf{aux}}$ leading to $\mathsf{root}^{(i)}_{\mathsf{aux}}$.
3. Compute $h = \mathsf{Hash}(\mathsf{root}^{(0)}_{\mathsf{aux}}, \ldots, \mathsf{root}^{(M-1)}_{\mathsf{aux}})$.
4. For all $i = 0, \ldots, M-1$:
   i. Perform commitment steps II.2. – II.4. from $\mathcal{P}$ to get $\mathsf{c}^{(i)}$.
5. Build tree T_{c} leading to $\mathsf{root}_{\mathsf{c}}$.
6. Send h and $\mathsf{root}_{\mathsf{c}}$ to verifier.

II. Challenge (Verifier)

Input: -

1. Sample uniform random $S \subseteq \{0, \ldots, M-1\}$ with $|S| = s$.
2. For all $j \in S$:
   i. Sample uniform random $z^{(j)} \in \mathbb{F}_q$.

3. Send S and $\{z^{(j)}\}_{j\in S}$ to prover.

III. Response (Prover)

Input: $S, \{z^{(j)}\}_{j\in S}$ and $\{\mathsf{seed}^{(j)}\}_{j\in S}$.

1. For all $j \in S$:
   i. Use $z^{(j)}$ and $\mathsf{seed}^{(j)}$ to compute response $\mathsf{rsp}^{(j)}$ for instance j.
2. Send $\{\mathsf{rsp}^{(j)}\}_{j\in S}, \{\mathsf{path}^{(j)}_{\mathsf{aux}}\}_{j\in S}, \{\mathsf{path}^{(j)}_{\mathsf{c}}\}_{j\in S}$ and $\mathsf{path}_{\mathsf{seed}}$ to verifier.

IV. Verification (Verifier)

Input: $\mathbf{H}, \mathbf{s}, h, \mathsf{root}_{\mathsf{c}}, \{\mathsf{rsp}^{(j)}\}_{j\in S}, \{\mathsf{path}^{(j)}_{\mathsf{aux}}\}_{j\in S}, \{\mathsf{path}^{(j)}_{\mathsf{c}}\}_{j\in S}$ and $\mathsf{path}_{\mathsf{seed}}$.

1. For all $j \in S$:
   i. Use $\mathsf{rsp}^{(j)}$ to compute $\mathsf{c}^{(j)}$.
   ii. Use $\mathsf{path}^{(j)}_{\mathsf{c}}$ to verify equality to $\mathsf{root}_{\mathsf{c}}$.
2. For all $j \in S$:
   i. Use $\mathsf{rsp}^{(j)}$ to compute $\mathsf{c}^{(j)}_{z^{(j)}}$.
   ii. Use $\mathsf{path}^{(j)}_{\mathsf{aux}}$ to compute $\mathsf{root}^{(j)}_{\mathsf{aux}}$.
3. For all $j \notin S$:
   i. Use $\mathsf{path}_{\mathsf{seed}}$ to recover $\mathsf{seed}^{(j)}$,
   ii. Perform precomputation steps and compute $\mathsf{root}^{(j)}_{\mathsf{aux}}$.
4. Compute $\mathsf{Hash}(\mathsf{root}^{(0)}_{\mathsf{aux}}, \ldots, \mathsf{root}^{(M-1)}_{\mathsf{aux}})$ and verify equality to h.
5. Accept if all checks in steps 1.ii and 4. are successful, or reject otherwise.

---

5 Practical Instantiation

Having defined a ZKID, the next step towards a practical instantiation as a signature scheme is to apply the Fiat-Shamir transformation. In fact, this allows to securely convert an interactive scheme (identification) into a non-interactive one (signature). To be precise, the following theorem was proved in [3], stating the security of a generalized version of the Fiat-Shamir transformation.

Theorem 1. *Let $\mathcal{I}$ be a canonical identification protocol that is secure against impersonation under passive attacks. Let $\mathcal{S} = \mathsf{FS}(\mathcal{I})$ be the signature scheme obtained applying the Fiat-Shamir transformation to $\mathcal{I}$. Then, $\mathcal{S}$ is secure against chosen-message attacks in the random oracle model.*

The main idea is to replace the interaction with the verifier in the challenge step with an automated procedure. Namely, the prover can generate the challenge by himself, by computing it as a hash value of the message and commitment. The signature will consist of the commitment and the corresponding response. The verifier can then regenerate the challenge himself, and proceed with the same verification steps as in the identification protocol. A schematic description of the process is given in the Appendix.

Note that the security result in Theorem 1 is somewhat vague, as the exact result depends on the specific security notions defined for identification and signature schemes. In our case, we rely on the fact that the underlying identification scheme provides honest-verifier zero-knowledge, with negligible soundness error, to achieve EUF-CMA security (see for example [21]). Moreover, note that, as per theorem statement, security depends on the hash function being modeled as a random oracle. This could, in principle, generate doubts about whether such a scheme would still be secure in the scenario that considers an adversary equipped with quantum capabilities. However, following some recent works [16, 22], we are able to claim that applying Fiat-Shamir to our identification scheme is enough to produce a signature scheme whose EUF-CMA security is preserved in the Quantum Random Oracle Model (QROM). In fact, as argued in [11, Theorem 3], it is possible to see that the tools used in our scheme satisfy the *collapsing* property, which is necessary to achieve existential unforgeability in the QROM. We refer the reader to [16] for a fully-detailed argument.

5.1 Parameters

We now move on to the discussion about practical parameter choices. We start by summarizing some important facts about SDP.

As a first consideration, note that, once one fixes the code parameters (i.e., the values of q, k and n), the difficulty to solve SDP depends heavily on the weight w of the searched solution. Generically, hard instances are those in which the ratio w/n is outside the range $[\frac{q-1}{q}(1-\frac{k}{n}); \frac{q-1}{q}+\frac{k}{nq}]$ (for a detailed discussion about this topic, we refer the interested reader to [15, Section 3]). Roughly speaking, the problem is hard when the weight of the solution is either high or low: in the first case SDP admits multiple solutions, while in the latter case we essentially expect to have a single solution. In this paper we will consider the low weight regime: as it is essentially folklore in coding theory, for random codes the hardest instances are obtained when w is close to the Gilbert-Varshamov (GV) distance, which is defined as

$$d(q,n,k) = \max\left\{ d \in \mathbb{N} \;\middle|\; \sum_{j=0}^{d-1} \binom{n}{j} (q-1)^j \leq q^{n-k} \right\}.$$

In such a regime, the best SDP solvers are known as ISD algorithms, as we mentioned in Section 2.1. Introduced by Prange in 1962 [26], ISD techniques have been characterized by a significant interest along the years, which has ultimately led to a strong confidence about their performance. In particular, for the case of non-binary codes, the state of the art is represented by Peters' algorithm [25], proposed in 2010 and still unbeaten in practice. In our scheme we will always set $w = d(q,n,k)$ and, to guarantee a security of λ bits, will choose parameters q, n and k so that the complexity resulting from Peters' ISD [25] is at least 2^λ.

Taking the above reasoning into account, we have devised several parameters sets, for different values of M. The resulting parameters are reported in Table 1, below.

M	s	q	n	k	w	Pk size (B)	Signature size (kB)
500	23	256	207	93	90	114	21.37
1000	19	256	207	93	90	114	17.97
5000	14	1024	189	85	86	130	14.95
10000	12	4096	176	79	83	145.5	13.91

Table 1: Parameters for the proposed instances, for several values of M.

We observe that our scheme offers an interesting trade-off between the signature size and the computational efficiency: increasing M leads to a significant reduction in the signature size, but this comes at the cost of an increase in the number of operations for signing and verification. Indeed, we expect the computational overhead to be dominated by the computation of keyed hash functions, whose number grows as Mq (these are required to obtained the M values of $\{\mathsf{aux}^{(i)}\}$). Optimization of the scheme can leverage a choice of efficient primitives for such short-input keyed hashes. In addition, we note that these hashes operate on independent inputs, so software and hardware performance can enjoy potential parallelization efficiency.

Remark 1. We note that, in order to decrease the algorithmic complexity of the scheme, one can reduce the size of the challenge space. Indeed, to select the challenges (and consequently, to prepare the $\mathsf{aux}^{(i)}$ values), one can use a set $C \subseteq \mathbb{F}_q$, of size $q' < q$. By doing so, the computation cost will decrease greatly. On the other hand the soundness error will also change, since in (2) we need to replace q with q', and thus the code parameters may have to change accordingly; however, this has very little impact on the communication cost, which will essentially remain unchanged (actually, it may become slightly smaller if representing the challenges requires fewer bits). To give an idea of how much the scheme's performance can benefit from this tweak, we present an example. For the last parameter set, where $M = 10000$, we can use a challenge space of size $q' = 3010$ instead of $q = 4096$. The signature size will not change, while the number of performed operations will get reduced approximately by 30%.

Remark 2. We would also like to point out that, while we have chosen all values of q that are powers of 2, this does not have to be the case. For the smallest parameter sets, for example, we estimate that a practical choice for the value of q would be either $q = 2^8$ or the prime $q = 251$. In both cases, the field arithmetic in the chosen field $\mathbb{F}_q$ could be implemented efficiently. In this context, note that Intel has recently included (as of microarchitecture codename "Ice Lake") the Galois Field New Instructions (GF-NI). These instructions (namely `VGF2P8AFFINEINVQB`, `VGF2P8AFFINEQB`, `VGF2P8MULB`) allow software to computed multiplications and inversions in $\mathbb{F}_{2^8}$ over the wide registers that are available with the AVX512 architectures.

References

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

[2] https://csrc.nist.gov/Presentations/2021/status-update-on-the-3rd-round. 2021.

[3] M. Abdalla et al. "From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security". In: *EUROCRYPT*. Springer. 2002, pp. 418–433.

[4] M. R. Albrecht et al. "Classic McEliece: conservative code-based cryptography". In: *NIST Post-Quantum Standardization, 3rd Round* (2021). URL: https://classic.mceliece.org/.

[5] N. Aragon et al. "BIKE: Bit Flipping Key Encapsulation". In: *NIST Post-Quantum Standardization, 3rd Round* (2021). URL: https://bikesuite.org/.

[6] J.-P. Aumasson et al. "SPHINCS$^+$". In: *NIST Post-Quantum Standardization, 3rd Round* (2021). URL: http://sphincs.org/.

[7] A. Barenghi et al. "LESS-FM: Fine-tuning Signatures from a Code-based Cryptographic Group Action." In: *PQCrypto 2021* (2021).

[8] S. Barg. "Some new NP-complete coding problems". In: *Problemy Peredachi Informatsii* 30.3 (1994), pp. 23–28. ISSN: 0555-2923.

[9] E. Berlekamp, R. McEliece, and H. van Tilborg. "On the inherent intractability of certain coding problems (Corresp.)" In: *IEEE Transactions on Information Theory* 24.3 (May 1978), pp. 384–386. ISSN: 0018-9448. DOI: 10.1109/TIT.1978.1055873.

[10] W. Beullens. "Sigma Protocols for MQ, PKP and SIS, and Fishy Signature Schemes". In: *EUROCRYPT (3)* 12107 (2020), pp. 183–211.

[11] W. Beullens. "Sigma protocols for MQ, PKP and SIS, and fishy signature schemes". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2020, pp. 183–211.

[12] J.-F. Biasse et al. "LESS is More: Code-Based Signatures Without Syndromes". In: *AFRICACRYPT*. Ed. by A. Nitaj and A. Youssef. Springer, 2020, pp. 45–65.

[13] P.-L. Cayrel, P. Véron, and S. M. El Yousfi Alaoui. "A zero-knowledge identification scheme based on the q-ary syndrome decoding problem". In: *Selected Areas in Cryptography*. Springer Berlin Heidelberg, 2011, pp. 171–186.

[14] N. T. Courtois, M. Finiasz, and N. Sendrier. "How to Achieve a McEliece-Based Digital Signature Scheme". In: *Advances in Cryptology - ASIACRYPT 2001, Lecture Notes in Computer Science* 2248 (2001), pp. 157–174.

[15] T. Debris-Alazard, N. Sendrier, and J.-P. Tillich. "Wave: A new family of trapdoor one-way preimage sampleable functions based on codes". In: *ASIACRYPT*. Springer. 2019, pp. 21–51.

[16] J. Don et al. "Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model". In: *CRYPTO 2019*, pp. 356–383.

[17] A. Fiat and A. Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *CRYPTO*. Springer. 1986, pp. 186–194.

[18] P. Gaborit and M. Girault. "Lightweight code-based identification and signature". In: *2007 IEEE International Symposium on Information Theory*. IEEE. 2007, pp. 191–195.

[19] A. Hülsing, J. Rijneveld, and F. Song. "Mitigating multi-target attacks in hash-based signatures". In: *Public-Key Cryptography–PKC 2016*. Springer, 2016, pp. 387–416.

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

[21] E. Kiltz, V. Lyubashevsky, and C. Schaffner. "A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pp. 552–586.

[22] Q. Liu and M. Zhandry. "Revisiting Post-quantum Fiat-Shamir". In: *Advances in Cryptology - CRYPTO 2019*. 2019, pp. 326–355.

[23] R. McEliece. "A Public-Key Cryptosystem Based On Algebraic Coding Theory". In: *Deep Space Network Progress Report* 44 (Jan. 1978), pp. 114–116.

[24] C. A. Melchor et al. "HQC: Hamming Quasi-Cyclic". In: *NIST Post-Quantum Standardization, 3rd Round* (2021). URL: `http://pqc-hqc.org/`.

[25] C. Peters. "Information-Set Decoding for Linear Codes over Fq". In: *Post-Quantum Cryptography*. Ed. by N. Sendrier. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 81–94.

[26] E. Prange. "The use of information sets in decoding cyclic codes". In: *IRE Trans. Inf. Theory* 8.5 (Sept. 1962), pp. 5–9.

[27] P. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509.

[28] J. Stern. "A new identification scheme based on syndrome decoding". In: *Advances in Cryptology — CRYPTO' 93*. Ed. by D. R. Stinson. Springer Berlin Heidelberg, 1994, pp. 13–21.

[29] P. Véron. "Improved identification schemes based on error-correcting codes". In: *Applicable Algebra in Engineering, Communication and Computing* 8.1 (1997), pp. 57–69.

A The Fiat-Shamir Transformation

Below, we give a schematic representation of the Fiat-Shamir transformation, applied to a canonical identification scheme $\mathcal{I}$, where we indicate with ℓ the length of the challenge.

Public Data, Private Key, Public Key

Same as in $\mathcal{I}$, plus a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\ell$.

I. Signature (Signer)

Input: Public data, private key and message $\mathbf{m}$.

1. Generate commitment cmt as in $\mathcal{I}$.
2. Compute challenge $\mathsf{ch} = \mathsf{H}(\mathbf{m}, \mathsf{cmt})$.
3. Produce response rsp as in $\mathcal{I}$.
4. Output signature $\sigma = (\mathsf{cmt}, \mathsf{rsp})$.

II. Verification (Verifier)

Input: Public data, public key, message $\mathbf{m}$ and signature σ.

1. Parse σ as $(\mathsf{cmt}, \mathsf{rsp})$.
2. Compute challenge $\mathsf{ch} = \mathsf{H}(\mathbf{m}, \mathsf{cmt})$.
3. Perform verification as in $\mathcal{I}$.
4. Accept or reject accordingly.
