# RYDE: A Digital Signature Scheme based on Rank-Syndrome-Decoding Problem with MPCitH Paradigm

Loïc Bidoux[1], Jesús-Javier Chi-Domínguez[1], Thibauld Feneuil[2,3], Philippe Gaborit[4], Antoine Joux[5], Matthieu Rivain[3], and Adrien Vinçotte[4]

[1] Technology Innovation Institute, UAE
[2] Sorbonne Université, CNRS, INRIA, Institut de Mathématiques de Jussieu-Paris Rive Gauche, Ouragan, Paris, France
[3] CryptoExperts, Paris, France
[4] University of Limoges, France
[5] CISPA, Helmholtz Center for Information Security, Saarbrücken

**Abstract.** We present a signature scheme based on the Syndrome-Decoding problem in rank metric. It is a construction from multi-party computation (MPC), using a MPC protocol which is a slight improvement of the linearized-polynomial protocol used in [Fen22], allowing to obtain a zero-knowledge proof thanks to the MPCitH paradigm. We design two different zero-knowledge proofs exploiting this paradigm: the first, which reaches the lower communication costs, relies on additive secret sharings and uses the hypercube technique [AMGH$^+$22]; and the second relies on low-threshold linear secret sharings as proposed in [FR22]. These proofs of knowledge are transformed into signature schemes thanks to the Fiat-Shamir heuristic [FS86].

# Table of Contents

# 1 Introduction

RYDE is a digital signature scheme for which the security relies on the hardness to solve the (unstructured) Rank-Syndrome-Decoding problem, supposed to be quantum resistant. It is based on the MPCitH paradigm, which provides a generic way to transform a MPC protocol into a zero-knowledge proof [IKOS07]. Even if we obtain a scheme with good performance based on an additive MPC, we also present another scheme adapted from the previous one to threshold MPC. Signature schemes are then simply obtained using the Fiat-Shamir heuristic.

In Section 2, we begin by defining all cryptographic and mathematical notions necessary for understanding. In Section 3, we give a high-level overview of the signature scheme (and leave low-level instructions to the specifications document). In Section 4, we describe the chosen parameters and their associated performances. In Section 5, we detail security proofs of the schemes. In Section 6, we detail attacks against signatures designed with the Fiat-Shamir transform and attacks against the Rank-Syndrome-Decoding problem.

In our schemes, the MPC protocol is a slight improvement on the linearized polynomials protocol for Rank-Syndrome-Decoding described in [Fen22]. The Hypercube-MPCitH idea comes from [AMGH+22], while the Threshold-MPCitH one is from [FJR22].

# 2 Preliminary notions

## 2.1 Notations and conventions

Let $A$ a randomized algorithm. We write $y \leftarrow A(x)$ the output of the algorithm for the input $x$. If $S$ is a set, we write $x \xleftarrow{\$} S$ the uniform sampling of a random element $x$ in $S$. We write $x \xleftarrow{\$,s} S$ the pseudo-random sampling in $S$ with seed $s$.

We denote by $\mathbb{F}_q$ the finite field of order $q$. We use bold letters to denote vectors and matrices (for example, $\boldsymbol{u} \in \mathbb{F}_q^n$ and $u \in \mathbb{F}_q$).

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 $\lambda$ is denoted $\mathsf{negl}(\lambda)$ while a polynomial function in $\lambda$ 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 $\lambda$ are $(t, \varepsilon)$-*indistinguishable* (where $t$ and $\varepsilon$ are $\mathbb{N} \to \mathbb{R}$ functions) if, for any algorithm $\mathcal{A}$ running in time at most $t(\lambda)$ we have

$$\left| \Pr[\mathcal{A}^{D_\lambda}() = 1] - \Pr[\mathcal{A}^{E_\lambda}() = 1] \right| \leq \varepsilon(\lambda) \ ,$$

with $\mathcal{A}^{Dist}$ meaning that $\mathcal{A}$ has access to a sampling oracle of distribution $Dist$. The two distributions are said

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

## 2.2 Security notions

**2.2.1 Digital signature schemes** We begin by remembering basic definitions and notions of security about signature schemes.

**Definition 1 (Digital signature scheme).** *A digital signature scheme* $\mathsf{DSS}$ *with security parameter* $\lambda$ *is a triplet of polynomial time algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ *such that:*

- *The key-generation algorithm* $\mathsf{KeyGen}$ *is a probabilistic algorithm which outputs a pair of keys* $(\mathsf{pk}, \mathsf{sk})$*;*

– *The signing algorithm* $\mathsf{Sign}$*, possibly probabilistic, which takes as inputs a message* $m$ *to be signed and the secret key* $\mathsf{sk}$*, and outputs a signature* $\sigma$*;*

– *The verification algorithm* $\mathsf{Verif}$ *which takes as inputs the public key* $\mathsf{pk}$*, the message* $m$ *and its signature* $\sigma$*, and outputs a bit* $b$*. Output* 1 *indicate that the signature is considered as valid.*

A correct signature scheme verifies the following property: if $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$, for all messages $m$ signed by $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}, m)$, we have $1 \leftarrow \mathsf{Verif}(\mathsf{sk}, m, \sigma)$. This means that a signature which is correctly generated is always accepted (when using the right keys).

The standard security notion for digital signature schemes is existential unforgeability under adaptive chosen message attacks (EUF-CMA) is defined as follows:

**Definition 2 (EUF-CMA).** *We can define the following game* $G_{\text{EUF-CMA}}$ *where* $\mathsf{Sign}(\mathsf{sk}, \cdot)$ *is an oracle that sign a message* $m^*$*:*
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}()$
$(m, \sigma) \leftarrow \mathcal{A}^{Sign(\mathsf{sk},\cdot)}(pk)$

---

*returns* 1 *if* $\mathsf{Verif}(m, \sigma, \mathsf{pk}) = 1$ *and* $m$ *was not queried to* $\mathsf{Sign}(\mathsf{sk}, \cdot)$
*The signature scheme is EUF-CMA if, for every polynomial adversary* $\mathcal{A}$*,* $\mathsf{Pr}\big(G_{\text{EUF-CMA}}(\mathcal{A}) = 1\big)$ *is negligible.*

**2.2.2 Commitment schemes** The security of the signature relies on a commitment scheme, which allows to commit on a value while hiding it until a possible future opening. We consider two properties: the commitment reveals no information about what is committed (hiding property), and there is only one (computationally tractable) way to open the commitment (binding property).

A commitment scheme takes as input any element $x$ (the value to commit) and a random tape $\rho \in \{0,1\}^\lambda$. When the secret value $x$ is revealed, the commitment $\mathsf{cmt} = \mathsf{Com}(x, \rho)$ can be verified by revealing $\rho$, which prove that $x$ has not been modified in the meantime.

**Definition 3 (Hiding).** *A commitment scheme* $\mathsf{Com}$ *is computationally hiding if, for every* $m_0, m_1$*, the distributions of*

$$\big\{\mathsf{Com}(m_0, \rho), \rho \xleftarrow{\$} \{0,1\}^\lambda\big\} \text{ and } \big\{\mathsf{Com}(m_1, \rho), \rho \xleftarrow{\$} \{0,1\}^\lambda\big\}$$

*are computationally indistinguishable.*

**Definition 4 (Binding).** *A commitment scheme Com is computationally biding if, for every PPT algorithm* $\mathcal{A}$*, we have:*

$$\mathsf{Pr}\big(\mathsf{Com}(m, \rho) = \mathsf{Com}(m', \rho') \cap m \neq m', (m, \rho, m', \rho') \leftarrow \mathcal{A}\big) < \mu(\lambda)$$

*where* $\mu$ *is a negligible function.*

### 2.2.3 Pseudo-random generators

**Definition 5 (Pseudo-random Generators).** *Let* $G : \{0,1\}^* \to \{0,1\}^*$*,* $\ell$ *a polynomial such that* $G(s) \in \{0,1\}^{\ell(\lambda)}$ *for* $s \in \{0,1\}^\lambda$*.* $G$ *is a* $(t, \epsilon)$*-secure pseudo-random generator if:*

– $\ell(\lambda) > \lambda$

– *The distributions* $\{G(s) |\, s \leftarrow \{0,1\}^\lambda\}$ *and* $\{r \mid r \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}\}$ *are* $(t, \epsilon)$*-indistinguishable*

In our constructions, we need *puncturable pseudo-random functions* (puncturable PRF). Such a primitive produces $N$ pseudo-random values and provides a way to reveal all the values but one. A standard construction of puncturable PRFs can be derived from the tree-based construction of [GGM86], usually denoted as the GGM construction. The idea is to use a tree PRG in which one uses a pseudorandom generator to expand a root seed into $N$ subseeds in a structured way. This can be illustrated quite easily with the following figure:

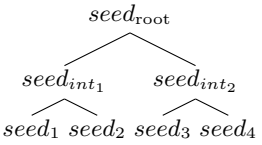


Fig. 1: Example of GGM tree

Now, imagine one is looking to reveal $seed_1$, $seed_3$, and $seed_4$, and hide $seed_2$. Then, all he has to do is reveal $seed_{int_2}$ and $seed_1$. It is impossible to retrieve $seed_2$, as we don't know the previous seed, but it is possible to retrieve the others. In this small example, we took $N = 4$. This is especially interesting as, in the additive-based MPCitH transformation we will see later, we reveal $N - 1$ leaves. This means that we can do this operation by revealing only $\log_2 N$ leaves instead of $N - 1$. More generally, for a GGM tree with $N$ leaves and given a subset $I \subset [1, N]$, it is possible to reveal all the leaves but the ones in $I$ by revealing at most $|I| \log_2(\frac{N}{|I|})$ nodes.

**2.2.4 Merkle trees** A collision-resistant hash function (that we will note $\mathsf{H}_M$) can be used to build a *Merkle Tree*. Given inputs $v_1 \dots v_N$, we define $\mathsf{Merkle}(v_1 \dots v_N)$ as:

$$\mathsf{Merkle}(v_1 \dots v_N) = \begin{cases} \mathsf{H}_M(\mathsf{Merkle}(v_1 \dots v_{\frac{N}{2}}) \| \mathsf{Merkle}(v_{\frac{N}{2}+1} \dots v_N)) \text{ if } N > 1 \\ \mathsf{H}_M(v_1) \text{ if } N = 1 \end{cases} \tag{1}$$

A Merkle Tree is a special case of vector commitment scheme allowing efficient opening of a subset of the committed values $v_i$. Given $I \subset [1, N]$, it is possible to verify that the $(v_i)_{i \in I}$ were indeed used to build the Merkle Tree only by revealing at most $|I| \log_2 \left(\frac{N}{|I|}\right)$ hash values. To proceed, we just need to reveal the sibling paths of the $(v_i)_{i \in I}$. Those paths are called the authentication paths and are denoted $\mathsf{auth}((v_1 \dots v_N), I)$.

**2.2.5 Zero-knowledge proofs of knowledge** We define here the concept of proof of knowledge, following [AFK21] notations. Let $R$ an NP-relation. $(x, \omega) \in R$ is a statement-witness pair where $x$ is the statement and $\omega$ an associated witness. The set of valid witness for a statement $x$ is $R(x) = \{\omega \mid (x, \omega) \in R\}$. A prover $\mathcal{P}$ wants to use a proof of knowledge to convince a verifier $\mathcal{V}$ that he knows a witness $\omega$ for a statement $x$.

**Definition 6 (Proof of knowledge).** *A proof of knowledge for a relation $R$ with soundness $\epsilon$ is a two-party protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with a public statement $x$, where $\mathcal{P}$ wants to convince $\mathcal{V}$ that he knows $\omega$ such that $(x, \omega) \in R$.*
*Let us denote $\langle \mathcal{P}(x, \omega), \mathcal{V}(x) \rangle$ the transcript between $\mathcal{P}$ and $\mathcal{V}$. A proof of knowledge must be perfectly complete, i.e.:*

$$\mathsf{Pr}\big(\mathsf{Accept} \leftarrow \langle \mathcal{P}(x, \omega), \mathcal{V}(x) \rangle\big) = 1$$

*If there exists a polynomial time prover $\tilde{\mathcal{P}}$ such that:*

$$\tilde{\epsilon} = \mathsf{Pr}\big(\mathsf{Accept} \leftarrow \langle \tilde{\mathcal{P}}(x), \mathcal{V}(x) \rangle\big) > \epsilon$$

*then there exists an algorithm $\mathcal{E}$, called extractor, which given rewindable access to $\tilde{\mathcal{P}}$, outputs a valid witness $\omega' \in R(x)$ in polynomial time in terms of $(\lambda, \frac{1}{\tilde{\epsilon} - \epsilon})$ with probability at least $\frac{1}{2}$.*

We now introduce the notion of honest-verifier zero-knowledge for a proof of knowledge (PoK):

**Definition 7 (Honest-Verifier Zero-Knowledge).** *A PoK satisfies the Honest-Verifier Zero-Knowledge (HZVK) property if there exists a polynomial time simulator $\mathsf{Sim}$ that given as input a statement $x$ and random challenges $(\mathsf{ch}_1, ..., \mathsf{ch}_n)$, outputs a transcript $\{\mathsf{Sim}(x, \mathsf{ch}_1, ..., \mathsf{ch}_n), \mathcal{V}(x)\}$ which is computationally indistinguishable from the probability distribution of transcripts of honest executions between a prover $\mathcal{P}(x, w)$ and a verifier $\mathcal{V}(x)$.*

**2.2.6 Fiat-Shamir transform** The Fiat-Shamir transform is a generic process allowing to transform an HVZK proof of knowledge into a signature scheme. The main adaptation lies in the removal of the interaction in the protocol: one needs to pull the challenges in a pseudo-random (and verifiable) way to sign a message without the assistance of a verifier. Note that the protocol must be repeated several times to achieve the desired level of security. We note $\tau$ the number of repetitions. One will see below that there is an effective attack against such signatures and so, $\tau$ must be chosen carefully (not too small).

Let us describe this transformation for the case of a 5-round protocol. The prover begins as in the zero-knowledge proof by committing all the auxiliary information. At the end of the first step, the prover computes:

$$h_1 = \mathsf{H}_1(\mathsf{salt}, m, (h_0^{(e)})_{e\in[1,\tau]})$$

where $\mathsf{H}_1$ is an hash function, $h_0^{(e)}$ is the first-step commitment of the $e^{\text{th}}$ execution of the protocol, and $\mathsf{salt}$ a random value in $\{0,1\}^{2\lambda}$. The prover obtains the first challenge from $h_1$ by using a XOF (extendable-output function).

The second challenge is generated in a similar way: the prover computes an element $h_2$ thanks to an other hash function $\mathsf{H}_2$ and the other information computed during the step 3. The prover obtains the second challenge from $h_2$ by using a XOF.

The signature $\sigma$ therefore consists of sending:

- $\mathsf{salt}$ which allows to compute commitments and the Fiat-Shamir hashes,
- $h_1$ as commitment of the initial values,
- $h_2$ as hash of all responses of the first challenge,
- each response $\mathsf{rsp}$ of the second challenge.

The signature is:

$$\sigma = (\mathsf{salt}, h_1, h_2, (\mathsf{rsp}^{(e)})_{e\in[1,\tau]})$$

### 2.3 MPC for proofs of knowledge

**2.3.1 Multi-Party Computation** We recall here the formalism proposed by [FR22]. The sharing of a value $x$ into $N$ parties is denoted $([\![x]\!]_1, ..., [\![x]\!]_N)$, where $[\![x]\!]_i$ is the share of index $i$ for $i \in [1, N]$, and $[\![x]\!]_J = ([\![x]\!]_i)_{i\in J}$ is the subset of shares for $J \subset [1, N]$.

**Definition 8 (Threshold LSSS).** *Let $\mathbb{F}$ a finite field and $t$ an integer in $[1, N]$. A $(t, N)$-threshold Linear Secret Sharing Scheme is a method to share a secret $s \in \mathbb{F}$ into $N$ shares $[\![s]\!] = ([\![s]\!]_1, ..., [\![s]\!]_N) \in \mathbb{F}^N$ such that the secret can be rebuilt from any subset of $t$ shares, while the knowledge of any subset of $t-1$ shares (or less) gives no information on $s$.*

Formally, a $(t, N)-$threshold LSSS is a pair of algorithms:

$$\begin{cases} \mathsf{Share} : \mathbb{F} \times R \to \mathbb{F}^N \\ \mathsf{Reconstruct}_J : \mathbb{F}^t \to \mathbb{F} \end{cases}$$

where $R$ denotes some randomness space, and $\mathsf{Reconstruct}_J$ is indexed by a subset $J \subset [1, N]$ of $t$ elements. These algorithms must verify the following properties:

- **Correctness:** for every $x \in \mathbb{F}$, $r \in R$ and $J \in [1, N]$ such that $|J| = t$: let $[\![x]\!] = \mathsf{Share}(x, r)$. We have:

$$\mathsf{Reconstruct}_J([\![x]\!]_J) = x$$

– **Perfect** $(t-1)$**-privacy:** For every $s_0, s_1 \in \mathbb{F}, I \subset [1,N]$ with $|I| = t-1$, the two distributions:

$$\left\{ [\![s_0]\!]_I \mid \begin{array}{c} r \xleftarrow{\$} R \\ [\![s_0]\!]_{[1,N]} \longleftarrow \mathsf{Share}(s_0, r) \end{array} \right\} \text{ and } \left\{ [\![s_1]\!]_I \mid \begin{array}{c} r \xleftarrow{\$} R \\ [\![s_1]\!]_{[1,N]} \longleftarrow \mathsf{Share}(s_1, r) \end{array} \right\}$$

are perfectly indistinguishable.

– **Linearity:** for every $v, v' \in \mathbb{F}^t$, $\alpha \in \mathbb{F}$ and subset $J \subset [1,N]$ of $t$ elements:

$$\mathsf{Reconstruct}_J(\alpha v + v') = \alpha \mathsf{Reconstruct}_J(v) + \mathsf{Reconstruct}_J(v')$$

Below are the two specific secret sharing which are used in RYDE:

**Definition 9 (Additive secret sharing).** *An additive secret sharing over a finite field $\mathbb{F}$ is an $(N,N)$-threshold LSSS where $R = \mathbb{F}^{N-1}$. The* Share *algorithm is defined as*

$$\mathsf{Share} : (s, (r_1, ..., r_{N-1})) \to [\![s]\!] = \left( r_1, ..., r_{N-1}, s - \sum_{i=1}^{N-1} r_i \right)$$

*The* $\mathsf{Reconstruct}_{[1,N]}$ *algorithm consists in computing the sum of all the shares.*

**Definition 10 (Shamir's secret sharing).** *The Shamir's secret sharing over a finite field $\mathbb{F}$ is an $(\ell+1, N)$-threshold LSSS with $R = \mathbb{F}^\ell$. Let $e_1, ..., e_N$ be public distinct elements of $\mathbb{F}^*$. The* Share *algorithm is defined as follows:*

– *Build a polynomial:* $P(X) = s + \sum_{i=1}^{\ell} r_i X^i$

– *For each $i \in [1,N]$: compute* $[\![s]\!]_i = P(e_i)$

*The* $\mathsf{Reconstruct}_J$ *algorithm consists in recovering $P$ with polynomial interpolation from known evaluations $[\![s]\!]_J = (P(e_i))_{i \in J}$. The reconstructed secret is then given by the constant term $s$ of $P$.*

A multi-party computation protocol is an interactive protocol involving several parties whose objective is to jointly compute a function $f$ on the share $[\![x]\!]$ they received at the begin of the protocol, so that they each get a share of $[\![f(x)]\!]$ at the end of the execution. In the restricted MPC model described in [FR22], at each step the parties can perform one of the following actions:

– Receiving elements: it can be randomness sent by a random oracle, or the shares of an hint which depends on the witness $\omega$ and the previous elements sent.

– Computing: since the sharing is linear, the parties can perform linear transformations on the shared values.

– Broadcasting: it is sometimes necessary to open some values (which give no information about the witness $\omega$) to execute the protocol.

For application of the MPCitH paradigm, this MPC protocol is only required to be secure in the semi-honest setting: we suppose that all parties rightfully perform their computation. The view of a party is made up of all the information available to this party when executing the protocol: its input shares and the messages it receives from other parties.

**Proposition 1 ([FR22]).** *Given a $(\ell+1, N)$-threshold LSSS, the following property holds; for each $v \in \mathbb{F}^{\ell+1}$ and each subset $J \subset [1,N]$ of $\ell+1$ elements, there exists a unique sharing $[\![x]\!] \in \mathbb{F}^N$ such that $[\![x]\!]_J = v$, and such that for all $\mathcal{J} \subset [1,N]$ of $\ell+1$ elements:*

$$\mathsf{Reconstruct}_{\mathcal{J}}([\![x]\!]_{\mathcal{J}}) = \mathsf{Reconstruct}_J(v).$$

One deduces that there exists an algorithm $\mathsf{Expand}_J$ which returns the unique sharing from a subset $J$ of the shares. For example, in the Shamir's secret sharing, Expand builds the Lagrange polynomial from the known evaluations and outputs the image of each party's point.

**2.3.2 MPC-in-the-Head paradigm** The MPC-in-the-Head paradigm, introduced by [IKOS07], is a framework allowing to convert a generic secure multi-party computation (MPC) protocol to a zero-knowledge proof. Let us assume we have a $\ell$-private MPC protocol in which $N$ parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ securely and correctly evaluate a function $f$ on a secret input $x$. The verifier asks the prover to reveal the views of $\ell$ parties (which include input shares and communications with other parties). In this construction, the zero-knowledge property comes from the $\ell$-privacy of the MPC protocol.

In what follows, we assume that all the manipulated MPC protocols fit the model decribed in [FR22]: they use $(\ell + 1, N)$-threshold LSSS and only uses the operations decribed in the previous subsection.

We want to build a zero-knowledge proof of knowledge of a witness for a statement $x$ such that $(x, \omega) \in \mathcal{R}$. Assume that we have a MPC protocol which evaluates a function $f$ on the witness $\omega$ and has the security properties mentioned above. The protocol works as follows:

- each party receives as input a share $[\![\omega]\!]_i$, where $[\![\omega]\!]$ is a threshold LSSS sharing of $\omega$,
- the function $f$ outputs ACCEPT if $(x, \omega) \in \mathcal{R}$, REJECT otherwise,
- the view of $\ell$ parties gives no information about the witness $\omega$.

In our case, the parties take as input a linear sharing $[\![\omega]\!]$ of the secret $\omega$ (one share per party) and they compute one or several rounds in which they perform three types of actions:

**Receiving randomness:** the parties receive a random value $\epsilon$ from a randomness oracle $\mathcal{O}_R$. When calling this oracle, all the parties get the same random value $\epsilon$.

**Receiving hint:** the parties can receive a sharing $[\![\beta]\!]$ (one share per party) from a hint oracle $\mathcal{O}_H$. The hint $\beta$ can depend on the witness $w$ and the previous random values sampled from $\mathcal{O}_R$.

**Computing & broadcasting:** the parties can locally compute $[\![\alpha]\!] := [\![\varphi(v)]\!]$ from a sharing $[\![v]\!]$ where $\varphi$ is an $\mathbb{F}$-linear function, then broadcast all the shares $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ to publicly reconstruct $\alpha := \varphi(v)$. The function $\varphi$ can depend on the previous random values $\{\epsilon^i\}_i$ from $\mathcal{O}_R$ and on the previous broadcast values. One should note that in the case of additive sharing, a single party needs to add a constant.

To build the zero-knowledge proof, the prover begins by building a sharing $[\![\omega]\!]$ of $\omega$. The prover then simulates all the parties of the protocol until the computation of $[\![f(\omega)]\!]$ (which should correspond to a sharing of ACCEPT, since the prover is supposed to be honest), and sends a commitment of each party's view. The verifier then chooses $\ell$ parties and asks the prover to reveal their views. The verifier checks the computation of their commitments. Since only $\ell$ parties have been opened, the revealed views give no information about $\omega$.

A naive way for the prover to execute the protocol would be to simulate all parties, but each party computes LSSS sharing of all involved elements. It implies that for each $a$ in these elements, a sharing $[\![a]\!] = ([\![a]\!]_1, ..., [\![a]\!]_N)$ contains redundancy when $\ell < N - 1$. One deduces that it is necessary to perform the calculations of the associated MPC protocol for only a subset $S$ of $\ell + 1$ out of $N$ parties, and commit them will be enough to ensure the soundness of the proof. When $\ell + 1$ is small, emulating the MPC protocol is cheap and leads to good overall performances: it is the high-level idea in Threshold-MPCitH. This technique to save computation is not possible when we use the additive sharing (since $\ell = N - 1$). However, in the additive setting, [AMGH+22] showed that the prover only need to emulate $1 + \log_2 N$ parties, and not $N$.

**2.3.3 Soundness** Consider a malicious prover who want to convince the verifier that he knows a witness $\omega$ for the instance $x$ (although it does not). Two cases can happen:

- The challenge of the verifier are such that the protocol results in a false positive,
- The malicious prover cheats for some party $i \in [1, N]$ (and hence has inconsistent view w.r.t. the commitments).

Feneuil and Rivain describe in [FR22] a strategy allowing a malicious prover to convince the verifier with probability:

$$\frac{1}{\binom{N}{\ell}} + p_\eta \frac{\ell\,(N-\ell)}{\ell+1}$$

where $p_\eta$ is the false positive rate of the associated MPC protocol (*i.e.* is the probability that the MPC protocol output ACCEPT even if $(x,\omega) \notin \mathcal{R}$). In the additive case ($\ell = N-1$), it leads to a soundness error of

$$\frac{1}{N} + p_\eta \left(1 - \frac{1}{N}\right).$$

### 2.4 Technical lemmas

We present here technical lemmas that will be useful in future proofs. The proofs of these lemmas can be found in the original references.

**Lemma 1 (Splitting Lemma [PS00]).** *Let $A \subset X \times Y$ such that $\mathsf{Pr}[(x,y) \in A] \geq \epsilon$.*
*For any $\alpha < \epsilon$, define*

$$B = \left\{(x,y) \in X \times Y \mid \mathsf{Pr}_{y' \in Y}[(x,y') \in A] \geq \epsilon - \alpha\right\} \text{ and } \bar{B} = (X \times Y) \setminus B$$

*then:*

- $\mathsf{Pr}[B] \geq \alpha$
- $\forall (x,y) \in B, \mathsf{Pr}_{y' \in Y}[(x,y') \in A] \geq \epsilon - \alpha$
- $\mathsf{Pr}[B|A] \geq \dfrac{\alpha}{\epsilon}$

### 2.5 Mathematical background

**2.5.1 $q$-polynomials** We begin by recalling here the main properties of finite fields. Let $q$ a prime number and $m \in \mathbb{N}$. Remember that $\mathbb{F}_{q^m}$ is a linear space over $\mathbb{F}_q$. The $q$-polynomials, introduced in [Ore33], are very useful tools in finite vector spaces: in particular, they allow to characterize linear subspaces.

**Definition 11 (q-polynomial).** *A $q$-polynomial of $q$-degree $r$ is a polynomial in $\mathbb{F}_{q^m}[X]$ of the form:*

$$P(X) = X^{q^r} + \sum_{i=0}^{r-1} X^{q^i} p_i \qquad \text{with } p_i \in \mathbb{F}_{q^m}$$

**Proposition 2.** *Let $P$ a $q$-polynomial in $\mathbb{F}_{q^m}[X]$, $\alpha, \beta \in \mathbb{F}_q$, $x, y \in \mathbb{F}_{q^m}$. Then:*

$$P(\alpha x + \beta y) = \alpha P(x) + \beta P(y)$$

*Proof.* Comes directly from the linearity over $\mathbb{F}_q$ of the Frobenius endomorphism: $x \to x^q$.

One can then define a linear subspace in $\mathbb{F}_{q^m}$ with a $q$-polynomial.

**Proposition 3.** *The set of roots of a non zero $q$-polynomial of degree $r$ is a linear subspace of dimension lower than or equal to $r$.*

*Proof.* Let $P$ a $q$-polynomial of degree $r$. One can see $P$ as a linear application from $\mathbb{F}_{q^m}$ to $\mathbb{F}_q^m$. As an endomorphism kernel, the set of zeros forms a linear space.
Since it is a polynomial of degree $q^r$, $P$ has at most $q^r$ roots, which allows to obtain the majoration of the dimension.

**Proposition 4 ([Ore33]).** *Let $E$ a linear subspace of $\mathbb{F}_{q^m}$ of dimension $r \leq m$. Then there exists a unique monic $q$-polynomial of $q$-degree $r$ such that all element in $E$ is a root of $P$.*
*$P$ is called the annihilator polynomial of $E$.*

**2.5.2 Rank metric** While the Hamming metric – which counts the number of nonzero coordinates of a code word – is relevant in the case of a prime field (especially $\mathbb{F}_2$), the rank metric relies on an extension field $\mathbb{F}_{q^m}$.

**Definition 12.** *Let $\boldsymbol{x} = (x_1, ..., x_n) \in \mathbb{F}_{q^m}^n$, and $\mathcal{B} = (b_1, ..., b_m)$ an $\mathbb{F}_q$-basis of $\mathbb{F}_{q^m}$. Each coordinate $x_i$ can be associated with a vector $(x_{i,1}, ..., x_{i,m}) \in \mathbb{F}_q^m$ such that:*

$$x_i = \sum_{j=1}^{m} x_{i,j} b_j$$

*The matrix $\boldsymbol{M}(\boldsymbol{x}) = (x_{i,j})_{(i,j)\in[1,n]\times[1,n]}$ is called matrix associated to the vector $\boldsymbol{x}$.*
*The rank weight is defined as: $W_R(\boldsymbol{x}) = \mathsf{Rank}(\boldsymbol{M}(\boldsymbol{x}))$.*
*The distance between two vectors is given by: $d(\boldsymbol{x}, \boldsymbol{y}) = W_R(\boldsymbol{x} - \boldsymbol{y})$.*
*The support of $\boldsymbol{x}$ is the linear subspace of $\mathbb{F}_{q^m}$ generated by its coordinates: $\mathsf{Supp}(\boldsymbol{x}) = \langle x_1, ..., x_n \rangle$*

*Remark:* The rank weight and the support of a vector is independent of the choice of the basis. On the other hand, the rank weight of $\boldsymbol{x}$ is obviously equal to the dimension of its support.

It is clear that the metric takes on its full meaning in the case of field extensions: if we were to place ourselves on a prime field, any non zero vector would have weight exactly 1. It also follows that the rank metric is less discriminating since increased by the Hamming weight.

The number of possible $r$-dimensional supports in $\mathbb{F}_{q^m}$ is given by the gaussian binomial coefficient:

$$\begin{bmatrix} m \\ r \end{bmatrix}_q = \prod_{i=0}^{r-1} \frac{q^m - q^i}{q^r - q^i}$$

One often use the following approximation: $\begin{bmatrix} m \\ r \end{bmatrix}_q \approx q^{r(m-r)}$.

### 2.6 Rank Syndrome Decoding problem

We want to build a signature based on a standard difficult problem: the syndrome decoding. However, rather than using the Hamming metric, we use here the rank metric, which allows to build similar constructions, but with different properties. We will begin by recalling the fundamental definitions inherent to coding theory.

**Definition 13.** *A linear code $\mathcal{C}$ on a finite field $\mathbb{F}$ of dimension $k$ and length $n$ is a linear subspace of $\mathbb{F}^n$ of dimension $k$. The elements in $\mathcal{C}$ are called the code words. One says that $\mathcal{C}$ is a $[n, k]_{\mathbb{F}}$ code.*

Two types of matrices can define a code:

**Definition 14.** *Let $\mathcal{C}$ a $[n, k]_{\mathbb{F}}$ code. A matrix $\boldsymbol{G} \in \mathbb{F}^{k\times n}$ is a generator matrix of $\mathcal{C}$ if its rows form a basis of $\mathcal{C}$, or equivalently:*

$$\mathcal{C} = \{\boldsymbol{m}\boldsymbol{G}, \boldsymbol{m} \in \mathbb{F}^k\}$$

*A matrix $\boldsymbol{H} \in \mathbb{F}^{(n-k)\times n}$ is a parity check matrix of $\mathcal{C}$ if its rows form a basis of $\mathcal{C}^\perp$, or equivalently:*

$$\mathcal{C} = \{\boldsymbol{c} \in \mathbb{F}^n, \boldsymbol{H}\boldsymbol{c}^\intercal = \boldsymbol{0}^\intercal\}$$

*Remark:* In order to lighten the notations, we often omit the transpose symbol for the vertical vectors

As in the case of the Hamming metric with random codes, syndrome decoding is a hard problem in the rank metric:

**Definition 15.** *Let $\mathbb{F}_{q^m}$ be the finite field of size $q^m$. Let $(n, k, r)$ be positive integers such that $k \leq n$. Let $I \in \mathbb{F}_{q^m}^{(n-k)\times(n-k)}$ the identity matrix of dimension $n-k$ on $\mathbb{F}_{q^m}$. The Rank Syndrom Decoding (Rank-SD) problem with parameters $(q, m, n, k, r)$ is the following one:*
*Let $\boldsymbol{H}$, $\boldsymbol{x}$ and $\boldsymbol{y}$ such that:*

- *$\boldsymbol{H}'$ is uniformly sampled from $\mathbb{F}_{q^m}^{(n-k)\times k}$, and $\boldsymbol{H} = (\boldsymbol{I} \,||\, \boldsymbol{H}') \in \mathbb{F}_{q^m}^{(n-k)\times n}$*
- *$\boldsymbol{x}$ is uniformly sampled from $\{\boldsymbol{x} \in \mathbb{F}_{q^m}^n, W_R(\boldsymbol{x}) \leq r\}$*
- *$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}$*

*From $(\boldsymbol{H}, \boldsymbol{y})$, find $\boldsymbol{x}$.*

*Remark:* Choosing the parity-check matrix $\boldsymbol{H}$ in standard form does not decrease the hardness of the problem, because obtaining this form from a random matrix can be done in polynomial time. As we see below, constraining this form allows us to share only the first $k$ coordinates of $\boldsymbol{x}$.

As we see later, there are two main approaches to solve the Rank-SD problem. The first one relies on combinatorial arguments: it consists in guessing the support of the error. The second one relies on algebraic computations, which consist in putting the problem into polynomial equations, then solving them using a Gröbner basis algorithm.

## 3 Description of the signature scheme

Here we explain how to build our signature scheme from a MPC protocol checking the solution of a Rank-SD instance. The MPC-in-the-Head paradigm allows us to convert the protocol first into a zero-knowledge proof, then into a signature scheme using the Fiat-Shamir transform.

### 3.1 MPC rank checking protocol

From the property 4 in the previous section, one can deduce a polynomial characterization of the rank of a vector: for all $\boldsymbol{x} \in \mathbb{F}_{q^m}$, $W_R(\boldsymbol{x}) \leq r$ if and only if there exists a $q$-polynomial $L$ of $q$-degree $r$ such that for all $i \in [1, n]$: $L(x_i) = 0$. The following MPC protocol is an optimization of the one proposed in [Fen22].

We now describe a MPC protocol to check the maximum rank of a vector $\boldsymbol{x} = (x_1, ..., x_n) \in \mathbb{F}_{q^m}$. Let $U = \mathsf{Supp}(\boldsymbol{x}) = \langle x_1, ..., x_n \rangle$ the vector space generated by its coordinates, and $L_U$ the annihilator polynomial of $U$:

$$L_U = \prod_{u \in U} (X - u) \in \mathbb{F}_{q^m}[X]$$

$L_U$ is a $q$-polynomial, so it can be written as:

$$L_U(X) = \sum_{i=0}^{r-1} \beta_i X^{q^i} + X^{q^r}$$

Since the support is defined up to multiplication by a constant, one can always assume that it contains 1 (see [AMAB+20]) without impacting the security of the scheme. So, we get that 1 is a root of the linearized polynomial $L_U$, we can consequently deduce a relation between its coefficients, and send one fewer value. We can suppose that $\boldsymbol{\beta} = (\beta_1, ..., \beta_{r-1}) \in \mathbb{F}_{q^m}^{r-1}$ and $\beta_0 = -\sum_{i=1}^{r-1} \beta_i - 1$.

Rather than checking separately that each $x_i$ is a root of $L_U$, we batch all these checks by uniformly sampling $\gamma_1, ..., \gamma_n$ in an extension $\mathbb{F}_{q^{m \cdot \eta}}$ of $\mathbb{F}_{q^m}$ and checking that:

$$\sum_{j=1}^{n} \gamma_j L_U(x_j) = 0$$

If one of the $x_i$ is not a root of $L_U$, the equation above is satisfied only with probability $\frac{1}{q^{m\eta}}$. We have

$$\begin{aligned}
\sum_{j=1}^{n}\gamma_j L(x_j) &= \sum_{j=1}^{n}\gamma_j\left(\sum_{i=0}^{r-1}\beta_i x_j^{q^i} + x_j^{q^r}\right)\\
&= \sum_{j=1}^{n}\gamma_j x_j^{q^r} + \sum_{i=0}^{r-1}\beta_i\cdot\sum_{j=1}^{n}\gamma_j x_j^{q^i}\\
&= \sum_{j=1}^{n}\gamma_j x_j^{q^r} + \beta_0\sum_{j=1}^{n}\gamma_j x_j + \sum_{i=1}^{r-1}\beta_i\cdot\sum_{j=1}^{n}\gamma_j x_j^{q^i}\\
&= \sum_{j=1}^{n}\gamma_j x_j^{q^r} + \Big(-\sum_{i=1}^{r-1}\beta_i - 1\Big)\sum_{j=1}^{n}\gamma_j x_j + \sum_{i=1}^{r-1}\beta_i\cdot\sum_{j=1}^{n}\gamma_j x_j^{q^i}\\
&= \sum_{j=1}^{n}\gamma_j x_j^{q^r} + \Big(-\sum_{i=1}^{r-1}\beta_i\Big)\sum_{j=1}^{n}\gamma_j x_j + \sum_{i=1}^{r-1}\beta_i\cdot\sum_{j=1}^{n}\gamma_j x_j^{q^i} - \sum_{j=1}^{n}\gamma_j x_j\\
&= \sum_{j=1}^{n}\gamma_j(x_j^{q^r} - x_j) + \sum_{i=1}^{r-1}\beta_i\cdot\sum_{j=1}^{n}\gamma_j(x_j^{q^i} - x_j)
\end{aligned}$$

Defining $z = -\sum_{j=1}^{n}\gamma_j(x_j^{q^r} - x_j)$, and $\omega_i = \sum_{j=1}^{n}\gamma_j(x_j^{q^i} - x_j)$, proving the equation is equivalent to proving that:

$$z = \langle\beta, \omega\rangle$$

with $\beta$ the vector of coefficients of $L_U$. This can be checked in the same way as in the multiplication checking protocol from [BN20]. Therefore, it is necessary to introduce a random vector $\boldsymbol{a} \in \mathbb{F}_{q^{m\cdot\eta}}^{r}$ and $c = -\langle\boldsymbol{\beta}, \boldsymbol{a}\rangle \in \mathbb{F}_{q^{m\cdot\eta}}$.

Since we consider a Rank-SD instance with $\boldsymbol{H}$ in standard form $\boldsymbol{H} = (\boldsymbol{I} \,||\, \boldsymbol{H}')$, the secret solution $\boldsymbol{x} \in \mathbb{F}_{q^m}^{n}$ can be split it in two parts $\boldsymbol{x} = (\boldsymbol{x}_A \parallel \boldsymbol{x}_B)$ with $\boldsymbol{x}_A \in \mathbb{F}_{q^m}^{k}$ and $\boldsymbol{x}_B \in \mathbb{F}_{q^m}^{n-k}$, such that

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} \quad \Leftrightarrow \quad \boldsymbol{x}_A = \boldsymbol{y} - \boldsymbol{H}'\boldsymbol{x}_B \ .$$

For a given instance, $(\boldsymbol{H}, \boldsymbol{y})$, the secret solution can then be fully (and linearly) recovered from $\boldsymbol{x}_B$ by $\boldsymbol{x} = (\boldsymbol{y} - \boldsymbol{H}'\boldsymbol{x}_B \parallel \boldsymbol{x}_B)$.

The resulting MPC protocol, denoted $\Pi^{\eta}$, is presented in figure 2. In a high-level point of view, the difference between this MPC protocol and the one described in [Fen22] comes from the fact that we assume that 1 is in the support of $x$ to save communication. However, this difference does not impact the security analysis (*i.e.* the false-positive rate) of the protocol, see below.

**Proposition 5 ([Fen22]).** *If $W_R(\boldsymbol{x}) \leq r$, then the protocol $\Pi^{\eta}$ always accepts. If $W_R(\boldsymbol{x}) > r$, the protocol accepts with probability at most: $p_\eta = \frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}$.*
*We call $p_\eta$ the false positive rate of $\Pi^{\eta}$.*

*Proof.* The protocol takes place in two stages: from steps 1 to 4, it computes the values $z$ and $\omega$. From step 5, it checks that $z = \langle\boldsymbol{\beta}, \boldsymbol{\omega}\rangle$. If $W_R(\boldsymbol{x}) > r$, there exists $k \in [1, n]$ such that $L(x_k) \neq 0$.
There are two possibilities for the protocol to accept in this case:

- either $\sum_{j=1}^{n}\gamma_j L(x_j) = 0$, which occurs with probability $\frac{1}{q^{m\eta}}$;
- or $z \neq \langle\boldsymbol{\beta}, \boldsymbol{\omega}\rangle$ but the protocol still accepts, which occurs with probability $\frac{1}{q^{m\eta}}$ (according to the false-positive rate of the [BN20] multiplication checking protocol adapted for the matrix setting, see [Fen22] for details).

**Public values.** instances of a Rank-SD problem : $\boldsymbol{H} = (I\ \boldsymbol{H}') \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $\boldsymbol{y} \in \mathbb{F}_{q^m}^{n-k}$

**Inputs.** Each party takes a share of $[\![\boldsymbol{x}_B]\!]$ where $\boldsymbol{x} \in \mathbb{F}_{q^m}^n$. Let $U$ a $\mathbb{F}_q$-linear subspace of $\mathbb{F}_{q^m}$ of dimension $r$ which contains $x_1, \ldots, x_n$ and has for annihilator polynomial $L = \prod_{u\in U}(X-u)$. Each party takes a share of the following sharing as inputs : $[\![L]\!] = \sum_{i=1}^{r-1}[\![\beta_i]\!](X^{q^i} - X) + (X^{q^r} - X)$, $[\![\boldsymbol{a}]\!]$ where $\boldsymbol{a}$ is uniformly sampled from $\mathbb{F}_{q^{m\cdot\eta}}^{r-1}$ and $[\![c]\!]$ such that $c = -\langle\boldsymbol{\beta}, \boldsymbol{a}\rangle$.

**MPC Protocol :**

1. The parties sample uniformly random : $(\gamma_1, ..., \gamma_n, \varepsilon) \xleftarrow{\$} \mathbb{F}_{q^{m\cdot\eta}}^{n+1}$.
2. The parties locally compute $[\![\boldsymbol{x}_A]\!] = \boldsymbol{y} - \boldsymbol{H}'[\![\boldsymbol{x}_B]\!]$ and $[\![\boldsymbol{x}]\!] = ([\![\boldsymbol{x}_A]\!] \,||\, [\![\boldsymbol{x}_B]\!])$.
3. The parties locally compute $[\![z]\!] = -\sum_{j=1}^n \gamma_j \left([\![x_j]\!]^{q^r} - [\![x_j]\!]\right)$.
4. The parties locally compute for all $k$ from 1 to $r-1$ : $[\![\omega_k]\!] = \sum_{j=1}^n \gamma_j \left([\![x_j]\!]^{q^k} - [\![x_j]\!]\right)$.
5. The parties locally compute $[\![\boldsymbol{\alpha}]\!] = \varepsilon \cdot [\![\boldsymbol{\omega}]\!] + [\![\boldsymbol{a}]\!]$.
6. The parties open $\boldsymbol{\alpha}$.
7. The parties locally compute $[\![v]\!] = \varepsilon \cdot [\![z]\!] - \langle\boldsymbol{\alpha}, [\![\boldsymbol{\beta}]\!]\rangle - [\![c]\!]$.
8. The parties open $v$.
9. The parties output ACCEPT if $v = 0$, and reject otherwise.

Fig. 2: Protocol $\Pi^\eta$ : checking that an input is solution of an instance of Rank-SD problem

Since these two events are independent, the false positive rate is:

$$p_\eta = \frac{1}{q^{m\eta}} + \left(1 - \frac{1}{q^{m\eta}}\right)\frac{1}{q^{m\eta}} \ \leq\ \frac{2}{q^{m\eta}}.$$

### 3.2 Conversion to zero-knowledge proof optimized with the hypercube technique

We start by presenting the main idea of a proof of knowledge optimized with the hypercube technique as proposed by [AMGH+22]: sharing a secret in $N = 2^D$ additive shares, and arranging them in a geometrical structure. Let us consider an hypercube of dimension $D$, each dimension having 2 slots. These $2^D$ shares are called "leaf parties". They can be indexed in two ways: either by an integer in $[1, 2^D]$, or by a vector of coordinates $(i_1, ..., i_D) \in [1, 2]^D$.

Since we use an additive secret sharing, one can intertwine several executions where each party (what we will call a "main party") can be the sum of a subset of leaves, such that these subsets form a partition of the leaves. We build here $D$ intertwined protocols, and each of them has 2 main parties. For each dimension $d \in [1, D]$, each main share is the sum of the $2^{D-1}$ leaves which have the same value for $i_d$. The main parties will always be indexed by $(d, k) \in [1, D] \times [1, 2]$.

The leaf parties are the first to be generated. The process is the same as in the generation of additive shares in a standard MPCitH transformation: the prover uses a GGM tree (as a puncturable pseudo-random function) to derive $N$ seeds from a root seed $\mathsf{seed}$. From the $i^{\text{th}}$ leaf seed, each party $i \in [1, N-1]$ pseudo-randomly generates the input shares $[\![\boldsymbol{x}_B]\!]_i, [\![\boldsymbol{\beta}]\!]_i, [\![\boldsymbol{a}]\!]_i$ and $[\![c]\!]_i$ (see figure 2 for the definition of these shares) . The last share for $i = N$ is computed such that we obtain a valid secret sharing of the values (except for $\boldsymbol{a}$ which is random and therefore can be sample from $\mathsf{seed}_N$). Then the prover can derive the shares of the "main parties" by summing the shares according to the partitions. For example, the share of $c$ of the main party $(1, k)$ will be $\sum_{i_2,...,i_D} [\![c]\!]_{(k,i_2,...,i_D)}$, which is the sum of the $2^{D-1}$ leaf shares such that $i_1 = k$.

The resulting protocols are presented in figures 3, 4 and 5.

An important remark is that one could have chosen a number of main parties per dimension $n \neq 2$. The size of the signature depends on $N = n^D$, but not independently on $n$ and $D$. For example, choosing the MPC parameters $(n, D) = (2, 8)$ or $(n, D) = (4, 4)$ will have no impact on the signature size, since $n^D = 256$ in the both cases. However, it is in our interest to choose $n$ as small as possible: this will increase the number $D$ of interleaved MPC protocols, but reduce the number of computations (which are much more expensive). Consequently, we always choose $n = 2$ and select different dimensions $D$.

**Public data**: An instance of a Rank-SD problem: $\boldsymbol{H} = (I\ \boldsymbol{H}') \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $\boldsymbol{y} \in \mathbb{F}_{q^m}^{n-k}$

The prover wants to convince the verifier that he knows the solution $\boldsymbol{x} \in \mathbb{F}_{q^m}^{n}$ of the instance, i.e. such that $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}$.

**Step 1: Commitment**

1. The prover computes $\boldsymbol{\beta} = (\boldsymbol{\beta}_k)_{k\in[1,r-1]} \in \mathbb{F}_{q^m}^{r}$ the coefficients of the annihilator q-polynomial $L(X)$ associated to $\boldsymbol{x}$ such that $L(X) = \prod_{u\in U}(X-u) = (X^{q^r} - X) + \sum_{k=1}^{r-1} \boldsymbol{\beta}_k (X^{q^k} - X)$ and $\forall j \in [1,n], L(\boldsymbol{x}_j) = 0$

2. The prover samples a seed: $\mathsf{seed} \xleftarrow{\$} \{0,1\}^{\lambda}$.

3. The prover expand $\mathsf{seed}$ recursively using a GGM tree to obtain $N$ leaves and seeds $(\mathsf{seed}_{i'}, \rho_{i'})$.

4. For each $i \in [1, N-1]$:

   - The prover samples $(\llbracket \boldsymbol{x}_B \rrbracket_i, \llbracket \boldsymbol{\beta} \rrbracket_i, \llbracket \boldsymbol{a} \rrbracket_i \llbracket c \rrbracket_i) \xleftarrow{\$,\mathsf{seed}_i} PRG$ where PRG is a pseudo-random generator
   - $\mathsf{state}_i = \mathsf{seed}_i$

5. For the share $N$:

   - The prover samples $\llbracket \boldsymbol{a} \rrbracket_N \xleftarrow{\$,\mathsf{seed}_N} PRG$
   - The prover computes $\llbracket \boldsymbol{x}_B \rrbracket_N = \boldsymbol{x}_B - \sum_{i=1}^{N-1} \llbracket \boldsymbol{x}_B \rrbracket_i$, $\llbracket \boldsymbol{\beta} \rrbracket_N = \boldsymbol{\beta} - \sum_{i=1}^{N-1} \llbracket \boldsymbol{\beta} \rrbracket_i$ and $\llbracket c \rrbracket_N = -\langle \boldsymbol{a}, \boldsymbol{\beta} \rangle - \sum_{i=1}^{N-1} \llbracket c \rrbracket_i$
   - $\mathsf{state}_N = (\mathsf{seed}_N, \llbracket \boldsymbol{x}_B \rrbracket_N, \llbracket \boldsymbol{\beta} \rrbracket_N, \llbracket c \rrbracket_N)$

6. The prover computes the commitments: $\mathsf{cmt}_i = \mathsf{Com}\big(\mathsf{state}_i, \rho_i\big)$, where $\rho_i$ is a random tape sampled in $\{0,1\}^{\lambda}$ from $\mathsf{seed}_i$ for each $i \in [1, N]$.

7. The prover commits all the shares: $h_0 = \mathsf{H}(\mathsf{cmt}_1, \cdots, \mathsf{cmt}_N)$

8. The prover computes the main parties by summing all the leaves associated. Indexing each party by its coordinates on the hypercube: $i = (i_1, ..., i_D)$ where $i_k \in [1,2]$. For all main party index $p = (d,k) \in \{(1,1), ...(D,2)\}$: $\llbracket \boldsymbol{x}_B \rrbracket_{(d,k)} = \sum_{i:i_d=k} \llbracket \boldsymbol{x}_B \rrbracket_i$, $\llbracket \boldsymbol{\beta} \rrbracket_{(d,k)} = \sum_{i:i_d=k} \llbracket \boldsymbol{\beta} \rrbracket_i$, $\llbracket \boldsymbol{a} \rrbracket_{(d,k)} = \sum_{i:i_d=k} \llbracket \boldsymbol{a} \rrbracket_i$ and $\llbracket c \rrbracket_{(d,k)} = \sum_{i:i_d=k} \llbracket c \rrbracket_i$.

**Step 2: First Challenge**

9. The verifier randomly samples $\big((\gamma_j)_{j\in[1,n]}, \epsilon\big) \in \mathbb{F}_{q^m\cdot\eta}^{n} \times \mathbb{F}_{q^m\cdot\eta}$ and sends it to the prover.

**Step 3: First Response**

10. For each dimension $d \in [1, D]$, the prover executes the algorithm in Fig. 4 on the main parties of the current dimension: $(d,1), (d,2)$. He computes $(\llbracket \boldsymbol{\alpha} \rrbracket, \llbracket v \rrbracket, H_d) \longleftarrow \text{Algorithm4}\big((P_d, (\gamma_j)_{j\in[1,n]}, \epsilon\big)$

11. The prover commits the executions: $h_1 = \mathsf{H}(H_1, ..., H_D)$

**Step 4: Second Challenge**

12. The verifier randomly samples $i^* \in [1, N]$ and sends it to the prover.

**Step 4: Second Response and verification**

13. The prover sends to the verifier $\mathsf{cmt}_{i^*}$ and $\llbracket \boldsymbol{\alpha} \rrbracket_{i^*}$; and the sibling path of the share $i^*$ to retrieve $(\mathsf{state}_j, \rho_j)_{j\neq i^*}$.

14. The verifier can deduce all the leaves (with the exception of index $i^*$), and recover $h_0$ using the sibling path and $\mathsf{cmt}_{i^*}$.

15. For all dimension $d \in [1, D]$, the verifier runs the algorithm in Fig. 5 to get $\llbracket \boldsymbol{\alpha} \rrbracket$, $\llbracket v \rrbracket$ and $H_d$. He checks:

   - $v = 0$ for all the executions.
   - $\boldsymbol{\alpha}$ is the same for all $D$ main executions.
   - $h_1 = \mathsf{H}(H_1, ..., H_D)$

Fig. 3: Protocol optimized with the hypercube technique

On the other hand, it is not necessary to do computations on all the main shares. Once one has done the calculations on the main parties on a dimension $d \in [1, D]$, one knows the value of $\boldsymbol{\alpha}$ for the protocol. Therefore, one can do the computations for only $n-1$ parties of protocols associated to all other dimensions, since one can deduce the value of the last share. This makes a total of only $n + (n-1)(D-1)$ computed parties, and choosing $n = 2$ makes this value the smallest as possible with $N = n^D$ fixed. One obtains that it is necessary to do computations for only $D+1$ main parties.

**Inputs :** A set of shares for the dimension $k$: $([\![\boldsymbol{x}_B]\!], [\![\boldsymbol{\beta}]\!], [\![\boldsymbol{a}]\!], [\![c]\!])$ and a protocol challenge $\left((\gamma_j)_{j\in[1,n]}, \epsilon\right)$

**Outputs :** A set of shares $[\![v]\!]$ and a commitment $H$ of the execution

For each party $i \in [1, N]$ :

- Compute $[\![\boldsymbol{x}_A]\!]_i = \boldsymbol{y} - \boldsymbol{H}'[\![\boldsymbol{x}_B]\!]_i$ and $[\![\boldsymbol{x}]\!]_i = ([\![\boldsymbol{x}_A]\!]_i \,||\, [\![\boldsymbol{x}_B]\!]_i)$
- $[\![z]\!]_i = -\sum_{j=1}^{n} \gamma_j \left([\![x_j]\!]_i^{q^r} - [\![x_j]\!]_i\right)$
- Compute $[\![\omega_k]\!] = \sum_{j=1}^{n} \gamma_j \left([\![x_j]\!]^{q^k} - [\![x_j]\!]\right)$ for each $k \in [1, r-1]$
- Compute $[\![\boldsymbol{\alpha}]\!]_i = \epsilon \cdot [\![\boldsymbol{w}]\!]_i + [\![\boldsymbol{a}]\!]_i$ and reveals $\boldsymbol{\alpha}$
- Compute $[\![v]\!]_i = \epsilon \cdot [\![z]\!]_i - \langle \boldsymbol{\alpha}, [\![\boldsymbol{\beta}]\!]_i \rangle - [\![c]\!]_i$

The parties compute together : $H = \mathsf{H}([\![\boldsymbol{\alpha}]\!], [\![v]\!]))$

Fig. 4: Execution of the MPC protocol $\Pi^{\eta}$ on a set of main shares

**Inputs:** A leaf $i^*$ that one does not reveal, the main party shares $[\![\boldsymbol{\alpha}]\!]$ and $[\![v]\!]$ on which the hidden leaf $i^*$ depends on, all the other main parties shares $([\![\boldsymbol{x}_B]\!], [\![\boldsymbol{\beta}]\!], [\![\boldsymbol{a}]\!], [\![c]\!])$.

**Outputs:** $[\![\boldsymbol{\alpha}]\!], [\![v]\!]$ and a commitment $H$ of the execution

**For** all the main parties:

**If** the party $p = (d, k)$ contains the leaf $i^*$, i.e. $i_d^* = k$:

Set $[\![v]\!]_{i^*}$ such that $v = 0$

**Else**:

Do the same computations as in $\Pi^{\eta}$ to obtain the correct shares $[\![\boldsymbol{\alpha}]\!]_p$ and $[\![v]\!]_p$

Compute $H = \mathsf{H}([\![\boldsymbol{\alpha}]\!], [\![v]\!])$

Fig. 5: Check the executions of the MPC protocol $\Pi^{\eta}$ on a the main parties

**3.2.1 Application of the Fiat-Shamir transform** We deduce from the Fiat-Shamir transform a way to convert a proof of knowledge to a signature scheme, by transforming the proof in a non-interactive protocol. The signature scheme and its verification algorithm are depicted in figures 6 and 7. Note that we use a value $\mathsf{salt}$, as in [FJR22], in order to increase the security of the scheme as it reduces the probability of seeds collision.

*Remark:* At the step 11 of the signature scheme, sending the state of all shares $j \neq i^*$ consists on:

- If $i^* = N$, sending the sibling path associated to the share $N$;
- If $i^* \neq N$, sending the sibling path associated to the share $i^*$ and the shares $[\![\boldsymbol{x}_B^{(e)}]\!]_N$, $[\![\boldsymbol{\beta}^{(e)}]\!]_N$ and $[\![c^{(e)}]\!]_N$ to recover all the states $j \neq i^*$.

We show in Section 5.2 that this scheme guarantees good security against EUF-CMA attacks.

**Inputs**

- Secret key $\mathsf{sk} = \boldsymbol{x}_B$ with $\boldsymbol{x} = (\boldsymbol{x}_A \,||\, \boldsymbol{x}_B) \in \mathbb{F}_{q^m}^n$ such that $w_R(\boldsymbol{x}) = r$
- Public key $\mathsf{pk} = (\boldsymbol{H}, \boldsymbol{y})$ with $\boldsymbol{H} = (\boldsymbol{I}\ \boldsymbol{H}') \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $\boldsymbol{y} \in \mathbb{F}_{q^m}^{n-k}$ such that $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}$
- Message $m \in \{0,1\}^*$

**Step 1: Commitment**

1. Sample a random salt value $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\lambda}$
2. Compute $\boldsymbol{\beta} = (\boldsymbol{\beta}_k)_{k\in[1,r-1]} \in \mathbb{F}_{q^m}^r$ the coefficients of the annihilator q-polynomial $L(X)$ associated to $\boldsymbol{x}$ such that $L(X) = \prod_{u\in U}(X-u) = (X^{q^r} - X) + \sum_{k=1}^{r-1} \boldsymbol{\beta}_k (X^{q^k} - X)$ and $\forall j \in [1,n], L(\boldsymbol{x}_j) = 0$
3. For each iteration $e \in [1,\tau]$:
   - Sample a root seed : $\mathsf{seed}^{(e)} \leftarrow \{0,1\}^\lambda$
   - Expand $\mathsf{seed}^{(e)}$ recursively with a GGM tree to obtain $N$ seeds and randomness $(\mathsf{seed}_{i'}^{(e)}, \rho_{i'})$
   - **For** each $i \in [1,N]$ :
     - **If** $i' \neq N$ :
       - Sample $(\llbracket \boldsymbol{x}_B^{(e)} \rrbracket_i, \llbracket \boldsymbol{\beta}^{(e)} \rrbracket_i, \llbracket \boldsymbol{a}^{(e)} \rrbracket_i \llbracket c^{(e)} \rrbracket_i) \overset{\$,\mathsf{seed}_i^{(e)}}{\longleftarrow} PRG$ where PRG is a pseudo-random generator
       - $\mathsf{state}_i^{(e)} = \mathsf{seed}_i^{(e)}$
     - **Else** :
       - Sample $\llbracket \boldsymbol{a}^{(e)} \rrbracket_N \overset{\$,\mathsf{seed}_N^{(e)}}{\longleftarrow} PRG$
       - Compute $\llbracket \boldsymbol{x}_B^{(e)} \rrbracket_N = \boldsymbol{x}_B^{(e)} - \sum_{i=1}^{N-1} \llbracket \boldsymbol{x}_B^{(e)} \rrbracket_i$, $\llbracket \boldsymbol{\beta}^{(e)} \rrbracket_N = \boldsymbol{\beta}^{(e)} - \sum_{i=1}^{N-1} \llbracket \boldsymbol{\beta}^{(e)} \rrbracket_i$ and $\llbracket c^{(e)} \rrbracket_N = -\langle \boldsymbol{\alpha}^{(e)}, \boldsymbol{\beta}^{(e)} \rangle - \sum_{i=1}^{N-1} \llbracket c^{(e)} \rrbracket_i$
       - $\mathsf{state}_N^{(e)} = (\mathsf{seed}_N^{(e)}, \llbracket \boldsymbol{x}_B^{(e)} \rrbracket_N, \llbracket \boldsymbol{\beta}^{(e)} \rrbracket_N, \llbracket c^{(e)} \rrbracket_N)$
     - Compute $\mathsf{cmt}_i^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, \mathsf{state}_i^{(e)})$
4. For all main party index $p = (d,k) \in \{(1,1), ...(D,2)\}$ : $\llbracket \boldsymbol{x}_B^{(e)} \rrbracket_{(d,k)} = \sum_{i:i_d=k} \llbracket \boldsymbol{x}_B^{(e)} \rrbracket_i$, $\llbracket \boldsymbol{\beta}^{(e)} \rrbracket_{(d,k)} = \sum_{i:i_d=k} \llbracket \boldsymbol{\beta}^{(e)} \rrbracket_i$, $\llbracket \boldsymbol{a}^{(e)} \rrbracket_{(d,k)} = \sum_{i:i_d=k} \llbracket \boldsymbol{a}^{(e)} \rrbracket_i$ and $\llbracket c^{(e)} \rrbracket_{(d,k)} = \sum_{i:i_d=k} \llbracket c^{(e)} \rrbracket_i$.
5. Commit all the shares : $h_0^{(e)} = \mathsf{H}_1(\mathsf{salt}, e, \mathsf{cmt}_1^{(e)}, ..., \mathsf{cmt}_N^{(e)})$
6. Compute $h_1 = \mathsf{H}_2\left(\mathsf{salt}, m, h_0^{(1)}, ..., h_0^{(\tau)}\right)$

**Step 2: First Challenge**

7. Extend hash $\left((\gamma_j^{(e)})_{j\in[1,n]}, \epsilon^{(e)}\right)_{e\in[1,\tau]} \leftarrow PRG(h_1)$ where $\left((\gamma_j^{(e)})_{j\in[1,n]}, \epsilon^{(e)}\right)_{e\in[1,\tau]} \in (\mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^\eta)^\tau$

**Step 3: First Response**

8. For each iteration $e \in [1,\tau]$ :
   **For** each each dimension $k \in [1,D]$ :
   - Execute the algorithm in Fig. 4 to obtain $\llbracket \boldsymbol{\alpha}^{(e)} \rrbracket$, $\llbracket v^{(e)} \rrbracket$ and $H_k^{(e)}$
9. Compute $h_2 = \mathsf{H}_4(m, \mathsf{pk}, \mathsf{salt}, h_1, (H_1^{(e)}, ..., H_D^{(e)})_{e\in[1,\tau]})$

**Step 4: Second Challenge**

10. Extend hash $(i^{*(e)})_{e\in[1,\tau]} \leftarrow PRG(h_2)$ with $i^{*(e)} \in [1,N]$

**Step 5: Second Response**

11.For each iteration $e \in [1,\tau]$:

Compute $\mathsf{rsp}^{(e)} = \left((\mathsf{state}_j)_{j\neq i^*}, \mathsf{cmt}_{i^*(e)}, \llbracket \alpha^{(e)} \rrbracket_{i*(e)}\right)$

12. Output $\sigma = \left(\mathsf{salt}, h_1, h_2, (\mathsf{rsp}^{(e)})_{e\in[1,\tau]}\right)$

Fig. 6: Rank-SD Signature Scheme based on Hypercube MPCitH - Signature Algorithm

**Inputs**

- Public key $\mathsf{pk} = (\boldsymbol{H}, \boldsymbol{y})$ with $\boldsymbol{H} = (\boldsymbol{I}\ \boldsymbol{H}') \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $\boldsymbol{y} \in \mathbb{F}_{q^m}^{n-k}$ such that $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}$
- Message $m \in \{0,1\}^*$
- Signature $\sigma = (\mathsf{salt}, h_1, h_2, (\mathsf{rsp}^{(e)})_{e\in[1,\tau]})$, where $\mathsf{rsp}^{(e)} = \left((\mathsf{state}_j)_{j\neq i^*}, \mathsf{cmt}_{i^*(e)}, [\![\alpha^{(e)}]\!]_{i^*(e)}\right)$

**Step 1: Parse signature**

1. Sample $\left((\gamma_j^{(e)})_{j\in[1,n]}, \epsilon^{(e)}\right)_{e\in[1,\tau]} \xleftarrow{\$,h_1} (\mathbb{F}_{q^m}^n \times \mathbb{F}_{q^{m\cdot\eta}})^\tau$
2. Sample $i^{*(e)} \xleftarrow{\$,h_2} [1,N]$
3. Read $(\mathsf{state}_i^{(e)})_{i\in[1,N]})$ for each iteration $e \in [1,\tau]$.

**Step 2: Recompute $h_1$**

4. For each iteration $e \in [1,\tau]$:

   **For** each $i \neq i^{*(e)}$ :

   $\mathsf{cmt}_{i^*}^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, \mathsf{state}_{i^*}^{(e)})$

   $h_0^{(e)} = \mathsf{H}_1\left(\mathsf{salt}, e, \mathsf{cmt}_1^{(e)}, ..., \mathsf{cmt}_N^{(e)}\right)$

5. Compute $\bar{h}_1 = \mathsf{H}_2\left(m, \mathsf{salt}, h_0^{(1)}, ..., h_0^{(\tau)}\right)$

**Step 3: Recompute $h_2$**

6. For each iteration $e \in [1,\tau]$:

   - Simulate MPC protocol $\Pi$ on main parties
   - For each dimension $k \in [1,D]$ :

     Run the algorithm in Fig. 5 to get $\overline{H_k^{(e)}}$

7. Compute $\bar{h}_2 = \mathsf{H}_4\left(m, \mathsf{pk}, \mathsf{salt}, \bar{h}_1, \left(\overline{H_1^{(e)}}, ..., \overline{H_D^{(e)}}\right)_{e\in[1,\tau]}\right)$

**Step 4: Verify signature**

8. Return $(\bar{h}_1 = h_1) \wedge (\bar{h}_2 = h_2)$

Fig. 7: Rank-SD based signature scheme - Verification algorithm

### 3.3 Conversion to zero-knowledge proof using low-threshold linear secret sharing

**3.3.1 Application of the MPCitH paradigm** We explain below that the MPC-in-the-Head paradigm is a framework allowing to convert an MPC protocol with threshold linear secret sharing to a zero-knowledge proof. Like in the subsection 3.2, the explanations given below are correct only within the framework of the restricted model [FR22]. The principle of the process is that the prover emulates in his head the parties, and the verifier asks to the prover to reveal the views of $\ell$ parties out of $N$ (which includes shares and communications with other parties). It ensures the zero-knowledge property as long as the underlying MPC protocol relies on a $(\ell+1, N)$-threshold secret sharing.

To build such a proof, the prover begins by computing a sharing $[\![\omega]\!]$ of the witness $\omega$, solution of the Rank Syndrome-Decoding instance. The prover then simulates all the parties of the protocol until the computation of $[\![f(\omega)]\!]$ (which should correspond to a sharing of ACCEPT, since the prover is supposed to be honest), and sends a commitment of each party's view to show good faith. The verifier then chooses $\ell$ parties and asks to the prover to reveal their views. The verifier checks the computation of the their commitments. Since only $\ell$ parties have been opened, the revealed views give no information about $\omega$.

A naive way for the prover to execute the protocol would be to simulate all parties, but each party computes LSSS sharing of all involved elements. It implies that for each $x$ in these elements, a sharing $[\![x]\!] = ([\![x]\!]_1, ..., [\![x]\!]_N)$ contains redundancy. One deduces that it is necessary to perform the calculations of the associated MPC protocol for only a subset $S$ of $\ell+1$ out of $N$ parties, and commit them will be enough to ensure the soundness of the proof. The threshold protocol therefore ensures better performances for the

simulation of the MPC protocol than the additive case (especially when $\ell$ is small compared to $n$), since it is not necessary to compute all the parties.

*Remark:* This advantage on the full execution of the signing algorithm is partially mitigated by the performances of the symmetric cryptography primitives involved in the protocol. Threshold-MPCitH has better performances on the pseudo-random generation, but the commitment is faster (for the prover) with additive sharing. See [FR22] for more details.

The protocol resulting from these considerations can be found in figure 8.

**Public data** : An instance of a Rank-SD problem : $\boldsymbol{H} = (I\ \boldsymbol{H}') \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $\boldsymbol{y} \in \mathbb{F}_{q^m}^{n-k}$

The prover wants to convince the verifier that he knows the solution $\boldsymbol{x} \in \mathbb{F}_{q^m}^{n}$ of the instance, i.e. such that $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}$

**Step 1: Commitment**

1. The prover builds a set of shares into a $(\ell+1, N)-$threshold secret sharing of $\boldsymbol{x}_B$. The prover also build shares of the annihilator polynomial $L = \prod_{u\in U}(X-u) = X^{q^r} + \sum_{i=0}^{r-1} \beta_i X^{q^i}$, a vector $\boldsymbol{a}$ uniformly sampled from $\mathbb{F}_{q^{m\cdot\eta}}^{r}$, and $c = -\langle \boldsymbol{\beta}, \boldsymbol{a}\rangle$. Each party takes one share of the previous elements.

For each $i \in [1, N]$ : $\mathsf{state}_i = (\llbracket \boldsymbol{x}_B \rrbracket_i, \llbracket \boldsymbol{\beta} \rrbracket_i, \llbracket \boldsymbol{a} \rrbracket_i, \llbracket c \rrbracket_i)$

2. The prover computes the commitments : $\mathsf{cmt}_i = \mathsf{Com}\big(\mathsf{state}_i, \rho_i\big)$, where $\rho_i$ is a random tape sampled from $\{0,1\}^\lambda$ for each $i \in [1, N]$.

3. The prover computes and sends the Merkle tree root : $h_0 = \mathsf{Merkle}(\mathsf{cmt}_1, \cdots, \mathsf{cmt}_N)$

**Step 2: First Challenge**

4. The verifier sends $\big((\gamma_j)_{j\in[1,n]}, \epsilon\big) \in \mathbb{F}_{q^{m\cdot\eta}}^{n} \times \mathbb{F}_{q^{m\cdot\eta}}$ to the prover.

**Step 3: First Response**

5. The prover chooses a public subset $S$ of parties such that $|S| = \ell + 1$:

For each party $i \in S$ :

- Compute $\llbracket \boldsymbol{x}_A \rrbracket_i = \boldsymbol{y} - \boldsymbol{H}' \llbracket \boldsymbol{x}_B \rrbracket_i$ and $\llbracket \boldsymbol{x} \rrbracket_i = (\llbracket \boldsymbol{x}_A \rrbracket_i \,||\, \llbracket \boldsymbol{x}_B \rrbracket_i)$
- Compute $\llbracket z \rrbracket = -\sum_{j=1}^{n} \gamma_j \left( \llbracket x_j \rrbracket^{q^r} - \llbracket x_j \rrbracket \right)$.
- Compute for each $k$ from 1 to $r-1$ : $\llbracket \omega_k \rrbracket = \sum_{j=1}^{n} \gamma_j \left( \llbracket x_j \rrbracket^{q^k} - \llbracket x_j \rrbracket \right)$
- Compute $\llbracket \boldsymbol{\alpha} \rrbracket_i = \epsilon \cdot \llbracket \boldsymbol{w} \rrbracket_i + \llbracket \boldsymbol{a} \rrbracket_i$ and the parties reveals $\boldsymbol{\alpha}$
- Compute $\llbracket v \rrbracket_i = \epsilon \cdot \llbracket z \rrbracket_i - \langle \boldsymbol{\alpha}, \llbracket \boldsymbol{\beta} \rrbracket_i \rangle - \llbracket c \rrbracket_i$

6. The prover commits the values : $h_1 = \mathsf{H}\left( (\llbracket \boldsymbol{\alpha} \rrbracket_i, \llbracket v \rrbracket_i)_{i\in S} \right)$

**Step 4: Second Challenge**

7. The verifier sends a subset $I \subset [1, N]$ of parties such that $|I| = \ell$ to the prover

**Step 5: Second Response**

8. The prover sends $(\mathsf{state}_i, \rho_i)_{i\in I}$, and the authentication path to these commitments, to allow the verifier to rebuild the Merkle tree. The verifier deduces a value $\tilde{h}_0$.

9. The prover chooses a public party $i^* \in S \backslash I$, and sends $\llbracket \boldsymbol{\alpha} \rrbracket_{i^*}$ to allow the verifier to recover $\llbracket \boldsymbol{\alpha} \rrbracket = \mathsf{Reconstruct}_{I\cup\{i^*\}}(\llbracket \boldsymbol{\alpha} \rrbracket_{I\cup\{i^*\}})$

10. The verifier sets $\llbracket v \rrbracket_{i^*}$ such that $v = 0$.

11. The computes $\tilde{h}_1 = \mathsf{H}((\llbracket \boldsymbol{\alpha} \rrbracket_i, \llbracket v \rrbracket_i)_{i\in S})$.

12. The verifier outputs ACCEPT if $(\tilde{h}_0, \tilde{h}_1) = (h_0, h_1)$, REJECT otherwise.

Fig. 8: Zero-knowledge protocol, deduced from the MPC protocol with threshold

**3.3.2 Digital signature scheme from Rank-SD problem** In this section, one applies the Fiat-Shamir transformation to the previous interactive zero-knowledge proof in order to obtain a signature scheme. One transforms the zero-knowledge proof into a non-interactive protocol by deterministically sampling challenges thanks to a hash function. It must be run several times to achieve the desired level of security.

The resulting signature scheme is depicted in the figure 9.

**Inputs**

- Secret key $\mathsf{sk} = (\boldsymbol{x})$ with $\boldsymbol{x} = (\boldsymbol{x}_A \,||\, \boldsymbol{x}_B) \in \mathbb{F}_{q^m}^n$ such that $w_R(\boldsymbol{x}) = r$
- Public key $\mathsf{pk} = (\boldsymbol{H}, \boldsymbol{y})$ with $\boldsymbol{H} = (\boldsymbol{I}\ \boldsymbol{H}') \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $\boldsymbol{y} \in \mathbb{F}_{q^m}^{n-k}$ such that $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}$
- Message $m \in \{0,1\}^*$

**Step 1: Commitment**

1. Sample a random salt value $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\lambda}$
2. Compute $\boldsymbol{\beta} = (\boldsymbol{\beta}_k)_{k\in[1,r-1]} \in \mathbb{F}_{q^m}^r$ the coefficients of the annihilator q-polynomial $L(X)$ associated to $\boldsymbol{x}$ such that $L(X) = \prod_{u\in U}(X-u) = X^{q^r} + \sum_{k=1}^{r-1} \boldsymbol{\beta}_k (X^{q^k} - X)$ and $\forall j \in [1,n], L(\boldsymbol{x}_j) = 0$
3. For each iteration $e \in [1,\tau]$:
   - $\diamond$ Sample $\boldsymbol{a}^{(e)} \xleftarrow{\$} \mathbb{F}_{q^{m\cdot\eta}}^r$ and compute $c^{(e)}$ such that $c^{(e)} \in \mathbb{F}_{q^{m\cdot\eta}}$ and $c^{(e)} = -\langle \boldsymbol{\beta}, \boldsymbol{a}^{(e)} \rangle$
   - $\diamond$ For each party $i \in [1,N]$:
     - Compute the $(\ell+1, N)$-threshold LSSS $[\![\boldsymbol{x}_B^{(e)}]\!], [\![\boldsymbol{\beta}^{(e)}]\!], [\![\boldsymbol{a}^{(e)}]\!], [\![c^{(e)}]\!]$ of $\boldsymbol{x}_B, \boldsymbol{\beta}, \boldsymbol{a}^{(e)}$ and $c^{(e)}$
     - Compute $\mathsf{state}_i^{(e)} = ([\![\boldsymbol{x}_B^{(e)}]\!]_i, [\![\boldsymbol{\beta}^{(e)}]\!]_i, [\![\boldsymbol{a}^{(e)}]\!]_i, [\![c^{(e)}]\!]_i)$ and $\mathsf{cmt}_i^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{(e)})$
   - $\diamond$ Compute the Merkle tree root $h_0^{(e)} = \mathsf{Merkle}(\mathsf{cmt}_1^{(e)}, \cdots, \mathsf{cmt}_N^{(e)})$
4. Compute $h_1 = \mathsf{H}_1(m, \mathsf{pk}, \mathsf{salt}, (h_0^{(e)})_{e\in[1,\tau]})$

**Step 2: First Challenge**

5. Extend hash $\left((\gamma_j^{(e)})_{j\in[1,n]}, \epsilon^{(e)}\right)_{e\in[1,\tau]} \leftarrow PRG(h_1)$ where $\left((\gamma_j^{(e)})_{j\in[1,n]}, \epsilon^{(e)}\right)_{e\in[1,\tau]} \in (\mathbb{F}_{q^m}^n \times \mathbb{F}_{q^{m\cdot\eta}})^\tau$

**Step 3: First Response**

6. For each iteration $e \in [1,\tau]$:
   - $\diamond$ For each party $i \in S$ with $S$ a public subset of parties such that $|S| = \ell + 1$:
     - Compute $[\![\boldsymbol{x}_A^{(e)}]\!]_i = \boldsymbol{y} - \boldsymbol{H}'[\![\boldsymbol{x}_B^{(e)}]\!]_i$ and $[\![\boldsymbol{x}^{(e)}]\!]_i = ([\![\boldsymbol{x}_A^{(e)}]\!]_i \,||\, [\![\boldsymbol{x}_B^{(e)}]\!]_i)$
     - Compute $[\![z^{(e)}]\!]_i = -\sum_{j=1}^n \gamma_j \left([\![x_j^{(e)}]\!]_i^{q^r} - [\![x_j^{(e)}]\!]_i\right)$ and $\forall k \in [1, r-1], [\![w_k^{(e)}]\!]_i = \sum_{j=1}^n \left(\gamma_j [\![x_j^{(e)}]\!]_i^{q^k} - [\![x_j^{(e)}]\!]_i\right)$
     - Compute $[\![\boldsymbol{\alpha}^{(e)}]\!]_i = \epsilon^{(e)} \cdot [\![\boldsymbol{w}^{(e)}]\!]_i + [\![\boldsymbol{a}^{(e)}]\!]_i$
     - Compute $[\![v^{(e)}]\!]_i = \epsilon^{(e)} \cdot [\![z^{(e)}]\!]_i - \langle [\![\boldsymbol{\alpha}^{(e)}]\!]_i, [\![\boldsymbol{\beta}^{(e)}]\!]_i \rangle - [\![c^{(e)}]\!]_i$
7. Compute $h_2 = \mathsf{H}_2(m, \mathsf{pk}, \mathsf{salt}, h_1, ([\![\boldsymbol{\alpha}^{(e)}]\!]_i, [\![v^{(e)}]\!]_i)_{i\in S, e\in[1,\tau]})$

**Step 4: Second Challenge**

8. Extend hash $(I^{(e)})_{e\in[1,\tau]} \leftarrow PRG(h_2)$ where $(\{I \subset N \mid |I| = \ell\})^\tau$

**Step 5: Second Response**

9. For each iteration $e \in [1,\tau]$:
   - $\diamond$ Choose deterministically a party $i^{*(e)} \in S \setminus I^{(e)}$ and compute $[\![\boldsymbol{\alpha}^{(e)}]\!]_{i^{*(e)}}$
   - $\diamond$ Compute the authentication path $\mathsf{auth}^{(e)}$ associated to root $h_0^{(e)}$ and $(\mathsf{cmt}_i^{(e)})_{i\in I^{(e)}}$
   - $\diamond$ Compute $\mathsf{rsp}^{(e)} = \left(([\![\boldsymbol{x}_B^{(e)}]\!]_i, [\![\boldsymbol{\beta}^{(e)}]\!]_i, [\![\boldsymbol{a}^{(e)}]\!]_i, [\![c^{(e)}]\!]_i)_{i\in I^{(e)}}, \mathsf{auth}^{(e)}, [\![\boldsymbol{\alpha}^{(e)}]\!]_{i^{*(e)}}\right)$
10. Compute $\sigma = (\mathsf{salt}, h_1, h_2, (\mathsf{rsp}^{(e)})_{e\in[1,\tau]})$

Fig. 9: Rank based signature scheme with threshold - Signing algorithm

An entity that wants to verify the signature must then recompute the challenges using the hashes $h_1$ and $h_2$ provided in the signature. For each iteration $e \in [1,\tau]$, the prover sends $\mathsf{rsp}^{(e)}$, which includes all the shares belonging to the parties in $I$, and $[\![\boldsymbol{\alpha}^{(e)}]\!]_{i^{*(e)}}$ which allows to recompute $\boldsymbol{\alpha}^{(e)}$. As a result, this information is sufficient for the verifier to check the hashes. Since hash functions are assumed to be collision free, checking equality between the hashes is sufficient to assume the validity of the signature.

The verification algorithm is formally detailed in the figure 10.

As for the additive sharing case, we will prove in the section 5.2 that this scheme guarantees good security against EUF-CMA attack.

**Inputs**

- Public key $\mathsf{pk} = (\boldsymbol{H}, \boldsymbol{y})$ with $\boldsymbol{H} = (\boldsymbol{I}\ \boldsymbol{H}') \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $\boldsymbol{y} \in \mathbb{F}_{q^m}^{n-k}$ such that $\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}$
- Message $m \in \{0,1\}^*$
- Signature $\sigma = (\mathsf{salt}, h_1, h_2, (\bar{\mathsf{rsp}}^{(e)})_{e\in[1,\tau]})$

**Step 1: Parse signature**

1. Sample $\left((\gamma_j^{(e)})_{j\in[1,n]}, \epsilon^{(e)}\right)_{e\in[1,\tau]} \xleftarrow{\$,h_1} (\mathbb{F}_{q^m}^n \times \mathbb{F}_{q^m}^{\cdot\eta})^\tau$
2. Sample $(I^{(e)})_{e\in[1,\tau]} \xleftarrow{\$,h_2} (\{I \subset N \mid |I| = \ell\})^\tau$
3. For each iteration $e \in [1,\tau]$:
   - ◇ Choose deterministically $i^{*(e)}$ from $S \setminus I^{(e)}$
   - ◇ Parse $\bar{\mathsf{rsp}}^{(e)} := \left((\llbracket \bar{\boldsymbol{x}}_B^{(e)} \rrbracket_i, \llbracket \bar{\boldsymbol{\beta}}^{(e)} \rrbracket_i, \llbracket \bar{\boldsymbol{a}}^{(e)} \rrbracket_i, \llbracket \bar{c}^{(e)} \rrbracket_i)_{i\in\bar{I}^{(e)}}, \bar{\mathsf{auth}}^{(e)}, \llbracket \bar{\boldsymbol{\alpha}}^{(e)} \rrbracket_{i^{*(e)}}\right)$

**Step 1: Recompute $\boldsymbol{h}_1$**

4. For each iteration $e \in [1,\tau]$:
   - ◇ For each party $i \in I^{(e)}$:
     - Compute $\bar{\mathsf{state}}_i^{(e)} = (\llbracket \bar{\boldsymbol{x}}_B^{(e)} \rrbracket_i, \llbracket \bar{\boldsymbol{\beta}}^{(e)} \rrbracket_i, \llbracket \bar{\boldsymbol{a}}^{(e)} \rrbracket_i, \llbracket \bar{c}^{(e)} \rrbracket_i)$ and $\bar{\mathsf{cmt}}_i^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, i, \bar{\mathsf{state}}_i^{(e)})$
   - ◇ Compute the Merkle tree root $\bar{h}_0^{(e)}$ from $(\bar{\mathsf{cmt}}_i^{(e)})_{i\in I^{(e)}}$ and $\bar{\mathsf{auth}}^{(e)}$
5. Compute $\bar{h}_1 = \mathsf{H}_1(m, \mathsf{pk}, \mathsf{salt}, (\bar{h}_0^{(e)})_{e\in[1,\tau]})$

**Step 2: Recompute $\boldsymbol{h}_2$**

6. For each iteration $e \in [1,\tau]$:
   - ◇ For each party $i \in I^{(e)}$:
     - Compute $\llbracket \bar{\boldsymbol{x}}_A^{(e)} \rrbracket_i = \boldsymbol{y} - \boldsymbol{H}' \llbracket \bar{\boldsymbol{x}}_B^{(e)} \rrbracket_i$ and $\llbracket \bar{\boldsymbol{x}}^{(e)} \rrbracket_i = (\llbracket \bar{\boldsymbol{x}}_A^{(e)} \rrbracket_i \,||\, \llbracket \bar{\boldsymbol{x}}_B^{(e)} \rrbracket_i)$
     - Compute $\llbracket \bar{z}^{(e)} \rrbracket_i = -\sum_{j=1}^n \gamma_j \left( \llbracket \bar{x}_j^{(e)} \rrbracket_i^{q^r} - \llbracket \bar{x}_j^{(e)} \rrbracket_i \right)$ and $\forall k \in [1, r-1], \llbracket \bar{w}_k^{(e)} \rrbracket_i = \sum_{j=1}^n \gamma_j \left( \llbracket \bar{x}_j^{(e)} \rrbracket_i^{q^k} - \llbracket \bar{x}_j^{(e)} \rrbracket_i \right)$
     - Compute $\llbracket \bar{\boldsymbol{\alpha}}^{(e)} \rrbracket_i = \epsilon^{(e)} \cdot \llbracket \bar{\boldsymbol{w}}^{(e)} \rrbracket_i + \llbracket \bar{\boldsymbol{a}}^{(e)} \rrbracket_i$
   - ◇ Reconstruct $\bar{\boldsymbol{\alpha}}^{(e)}$ and $(\llbracket \bar{\boldsymbol{\alpha}}^{(e)} \rrbracket_i)_{i\in S}$ from $(\llbracket \bar{\boldsymbol{\alpha}}^{(e)} \rrbracket_i)_{i\in I^{(e)}}$ and $\llbracket \bar{\boldsymbol{\alpha}}^{(e)} \rrbracket_{i^{*(e)}}$
7. For each iteration $e \in [1,\tau]$:
   - ◇ For each party $i \in I^{(e)}$:
     - Compute $\llbracket \bar{v}^{(e)} \rrbracket_i = \epsilon^{(e)} \cdot \llbracket \bar{z}^{(e)} \rrbracket_i - \langle \bar{\boldsymbol{\alpha}}^{(e)}, \llbracket \bar{\boldsymbol{\beta}}^{(e)} \rrbracket_i \rangle - \llbracket \bar{c}^{(e)} \rrbracket_i$
   - ◇ Reconstruct $(\llbracket \bar{v}^{(e)} \rrbracket_i)_{i\in S}$ from $(\llbracket \bar{v}^{(e)} \rrbracket_i)_{i\in I^{(e)}}$ and $\bar{v}^{(e)} = 0$
8. Compute $\bar{h}_2 = \mathsf{H}_2(m, \mathsf{pk}, \mathsf{salt}, \bar{h}_1, (\llbracket \bar{\boldsymbol{\alpha}}^{(e)} \rrbracket_i, \llbracket \bar{v}^{(e)} \rrbracket_i)_{i\in S, e\in[1,\tau]})$

**Step 5: Verify signature**

9. Return $(\bar{h}_1 = h_1) \wedge (\bar{h}_2 = h_2)$

Fig. 10: Rank based signature scheme with threshold - Verification algorithm

**3.3.3 Using Shamir secret sharing** In practice, we use the $(\ell+1)$-threshold Shamir's secret sharing scheme as LSSS. Thus the shares of a secret value $s \in \mathbb{F}_q$ are defined as $\llbracket s \rrbracket_i = P(e_i)$, where $P$ is a polynomial whose constant term is equal to $s$, and $e_1, ..., e_N \in \mathbb{F}_q$ are public non zero distinct points. However, since each share corresponds to the evaluation of a polynomial into a distinct point of $\mathbb{F}_q$, we get that the number $N$ of parties is upper bounded by $q$. To have interesting performances, it means that $q$ must be large (for example, $q = 256$).

It is possible to mitigate this constraint. Let us assume that we want to share values directly from $\mathbb{F}_{q^m}$ (instead of $\mathbb{F}_q$). In that case, $e_1, \ldots, e_N$ would belong to $\mathbb{F}_{q^m}$ and we would have $N \leq q^m$ instead of $N \leq q$. The only issue comes from the usage of the Frobenius endormorphism which is $\mathbb{F}_q$-linear and not $\mathbb{F}_{q^m}$-linear. We can remark that, from a sharing $\llbracket s \rrbracket$ of $s$, each party $i$ can obtain a sharing of $s^q$ by simply computing $\llbracket s \rrbracket_i^q$. However, the evaluation points of the obtained sharing and the encoding polynomial changed: they

become $e_1^q, ..., e_N^q$. Indeed,

$$\begin{aligned} P(X)^q &= \left( s + \sum_{i=1}^{\ell} r_i X^i \right)^q \\ &= s^q + \sum_{i=1}^{\ell} r_i^q (X^q)^i \\ &= \tilde{P}(X^q) \qquad \text{where} \quad \tilde{P} = s^q + \sum_{i=1}^{\ell} r_i^q X^i \end{aligned}$$

One can add two shares if and only if they have the same evaluation points. The optimized MPC protocol described used in figure 9 does not satisfy this property: we can not set the input sharings such that we have the guarantee that addition will be performed only on sharings with the same evaluation points. However, if we want to have a scheme relying on Threshold-MPCitH with a small $q$ (for example, with $q = 2$), we can use the MPC protocol proposed in [Fen22]. The latter satisfies the desired properties (see [Fen22, Appendix A] for details).

# 4 Parameter sets

## 4.1 Choice of parameters

The problem Rank Syndrome Decoding is parameterized by the following parameters:

- $q \in \mathbb{N}$ - the order of the base field on which the problem is based
- $m \in \mathbb{N}$ - the degree of the considered extension of the above field
- $n \in \mathbb{N}$ - the length of the code $\mathcal{C}$
- $k \in \mathbb{N}$ - the dimension of the code
- $r \in \mathbb{N}$ - the rank of the vector $\boldsymbol{x}$

The other parameters of the scheme are:

- $N \in \mathbb{N}$ - the number of parties simulated in the MPC protocol
- $\tau \in \mathbb{N}$ - the number of rounds in the signature
- $\eta \in \mathbb{N}$ - allowing to build $\mathbb{F}_{q^{m \cdot \eta}}$

In order to choose the parameters, we need to consider:

- The security of the Rank-SD instance, i.e, the complexity of the attacks on the chosen parameters.
- The security of the signature scheme, i.e, the cost of a forgery.
- The size of the signature

Since the protocol is more efficient as $N$ grows (up to some point), we need to take a larger value of $q$ in the threshold variant. To have a threshold scheme with $N = 256$ shares, we must take $q = 256$ since we necessarily have $N \leq q$ (in the Shamir Secret Sharing scheme, we can't have more shares than the number of elements in the base field; this means we can't have more parties than the cardinal of $\mathbb{F}_q$ in our case).

In additive sharing, we take the value of $q = 2$, since it is the most efficient and we have no constraints. We propose here two versions of the signature scheme with different values for $N$: we set $N = 256$ for a short version of the signature, and $N = 32$ for a fast one.

The choice of $(q, m, n, k, r)$ was then made in order to achieve the target security in regards to the attacks we present in subsection 6.2.

As for the choice of $\tau$ and $\eta$, we need to choose them such that our signature scheme resists to the forgery attack described by [KZ20], which we explain in subsection 6.1. Because of this, in the additive protocol optimized with the hypercube technique, we need to set $\tau$ such that:

$$\text{cost}_{\text{forge}} = \min_{0 \leq \tau' \leq \tau} \left\{ \frac{1}{\sum_{i=\tau'}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau - i}} + N^{\tau - \tau'} \right\} \tag{2}$$

is higher than $2^\lambda$, with $p = \frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}$. The value depends on $\tau$ and $\eta$, we then need to take them such that the signature is safe, and as small as possible, given the parameters of the Rank-SD instance.

In the case of the threshold protocol, this formula changes:

$$\text{cost}_{\text{forge}} = \min_{0 \leq \tau' \leq \tau} \left\{ \frac{1}{\sum_{i=\tau'}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau - i}} + \binom{N}{\ell}^{\tau - \tau'} \right\} \tag{3}$$

with $p = (\frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}) \cdot \binom{N}{\ell+1}$, see [FR22] for details.
In any case, as long as the cost of the forgery is high enough, we can just take the parameters for which the size of signature is the smallest.

We propose the following parameters for a security of $\lambda = 128$ bits:

– For the additive scheme: $(q, m, n, k, r) = (2, 31, 33, 15, 10)$

– For the threshold scheme with $\ell = 3$: $(q, m, n, k, r) = (256, 11, 12, 5, 5)$

– For the threshold scheme with $\ell = 3$ with small $q$: $(q, m, n, k, r) = (2, 31, 29, 14, 10)$

### 4.2 Signature and key sizes

**4.2.1 Additive MPC** We must send for each $e \in [1, \tau]$ the following elements: $\mathsf{cmt}_{i^*} \in \{0,1\}^{2\lambda}$ and $(\mathsf{state}_i^{(e)})_{i \neq i^*}$ (which consists in seeds in $\{0,1\}^{2\lambda}$ or the following elements for one of them: $[\![\boldsymbol{x}_B]\!] \in \mathbb{F}_{q^m}^k$, $[\![\boldsymbol{\alpha}]\!] \in \mathbb{F}_{q^{m\cdot\eta}}^{r-1}$, $[\![\boldsymbol{\beta}]\!] \in \mathbb{F}_{q^m}^{r-1}$, $[\![c]\!] \in \mathbb{F}_{q^{m\cdot\eta}}$); and $(\mathsf{salt}|h_1|h_2) \in \{0,1\}^{6\lambda}$.

Note also that for each $e \in [1, \tau]$, we do not send all the $N-1$ seeds $(\mathsf{state}_i^{(e)})$, since we use a tree-PRG (noted TPRG in the figures) to generate them: starting from the root seed, each seed is recursively expanded in two child seeds until we obtain $N$ seeds. We obtain a tree of depth $\log_2 N$, and we only reveal the $\log_2 N$ sibling nodes of the root-leaf path of the hidden leaf $i^*$ to reveal all the leaves except $i^*$.

We can now compute the communication cost of the protocol (in bits):

$$|\sigma| = \underbrace{6\lambda}_{\mathsf{salt},\, h1,\, h2} + \tau \cdot \left( \Big( \underbrace{(r-1)m\eta}_{\boldsymbol{\alpha}} + \underbrace{km}_{\boldsymbol{x}_B} + \underbrace{(r-1)m}_{\boldsymbol{\beta}} + \underbrace{m\eta}_{c} \Big) \cdot \log_2 q + \underbrace{2\lambda + (\log_2 N)\lambda}_{\text{additive MPCitH}} \right)$$

Parameters are chosen such the Rank-SD problem is secure against the existing attacks (see section 6.2). We recall that the security levels of the signature I, III and V correspond respectively to the security of `AES-128`, `AES-192` and `AES-256`. Tables 1 and 2 give the theoretical signature size for different set of secure parameters in the case of additive shares:

| NIST security level | $q$ | $m$ | $n$ | $k$ | $r$ | $N$ | $\eta$ | $\tau$ | $\mathsf{pk}$ | $\sigma$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 31 | 33 | 15 | 10 | 256 | 1 | 20 | 0.1 kB | 5.9 kB |
| 3 | 2 | 37 | 41 | 18 | 13 | 256 | 1 | 29 | 0.1 kB | 12.9 kB |
| 5 | 2 | 43 | 47 | 18 | 17 | 256 | 1 | 38 | 0.1 kB | 22.8 kB |

Table 1: Parameters for additive MPC on hypercube, short signature

| NIST security level | $q$ | $m$ | $n$ | $k$ | $r$ | $N$ | $\eta$ | $\tau$ | $\mathsf{pk}$ | $\sigma$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 31 | 33 | 15 | 10 | 32 | 1 | 30 | 0.1 kB | 7.4 kB |
| 3 | 2 | 37 | 41 | 18 | 13 | 32 | 1 | 44 | 0.1 kB | 16.4 kB |
| 5 | 2 | 43 | 47 | 18 | 17 | 32 | 1 | 58 | 0.1 kB | 29.1 kB |

Table 2: Parameters for additive MPC on hypercube, fast signature

**4.2.2 Threshold MPC** In the threshold case, the process is similar to the additive one above: we must send for each $e \in [1, \tau]$ the following elements: $h \in \{0,1\}^{2\lambda}$ (an hash used) and $\ell$ times all the elements$(\mathsf{state}_i^{(e)})_{i \neq i^*}$; and one time $(\mathsf{salt}|h_1|h_2) \in \{0,1\}^{6\lambda}$.

The signature size of the scheme is given by

$$|\sigma| \leq \underbrace{6\lambda}_{\mathsf{salt},h1,h2} + \tau \cdot \Big( \ell \cdot \big( \underbrace{km}_{\boldsymbol{x}_B} + \underbrace{(r-1)m\eta}_{\boldsymbol{\alpha}} + \underbrace{(r-1)m}_{\boldsymbol{\beta}} + \underbrace{rm\eta}_{\boldsymbol{a} \text{ and } c} \big) \cdot \log_2(q) + \underbrace{2\lambda \cdot \ell \log_2(\frac{N}{\ell})}_{\text{Threshold MPCitH}} \Big)$$

Let us recall that the scheme requires that $N \leq q$. If we want to use a small $q$, we need to replace our MPC protocol with the one described in [Fen22] (see section 3.3.3 for details). In that case, the signature size is given by

$$|\sigma| \leq \underbrace{6\lambda}_{\mathsf{salt},h1,h2} + \tau \cdot \Big( \ell \cdot \big( \underbrace{km}_{\boldsymbol{x}_B} + \underbrace{rm\eta}_{\boldsymbol{\alpha}} + \underbrace{(r-1)m}_{\boldsymbol{\beta}} + \underbrace{rm\eta}_{\boldsymbol{a} \text{ and } c} \big) \cdot \log_2(q) + \underbrace{2\lambda \cdot \ell \log_2(\frac{N}{\ell})}_{\text{Threshold MPCitH}} \Big)$$

Tables 3 and 4 give the theoretical signature size for different set of secure parameters in the case of threshold shares for $\ell = 3$ and $\ell = 1$:

| NIST security level | $q$ | $m$ | $n$ | $k$ | $r$ | $N$ | $\eta$ | $\tau$ | $\mathsf{pk}$ | $\sigma$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 256 | 11 | 12 | 5 | 5 | 256 | 2 | 6 | 0.1 kB | 8.2 kB |
| 3 | 256 | 13 | 17 | 7 | 6 | 256 | 1 | 11 | 0.1 kB | 18.3 kB |
| 5 | 256 | 17 | 17 | 7 | 7 | 256 | 3 | 14 | 0.1 kB | 32.5 kB |

Table 3: Parameters for threshold with $\ell = 3$, $q = 256$

| NIST security level | $q$ | $m$ | $n$ | $k$ | $r$ | $N$ | $\eta$ | $\tau$ | $\mathsf{pk}$ | $\sigma$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 31 | 33 | 15 | 10 | 256 | 2 | 18 | 0.1 kB | 9.3 kB |
| 3 | 2 | 37 | 41 | 18 | 13 | 256 | 2 | 27 | 0.1 kB | 21.4 kB |
| 5 | 2 | 43 | 47 | 18 | 17 | 256 | 2 | 35 | 0.1 kB | 34.8 kB |

Table 4: Parameters for threshold with $\ell = 1$, $q = 2$

# 5 Security analysis

## 5.1 Security proof for the proof of knowledge

**5.1.1 Additive MPC** We prove here that our protocol fulfils the three required properties: completeness, soundness and zero-knowledge. The proofs of this section are close to those in [AMGH$^+$22] (which are themselves close to [FJR22]) due to the same structure of hypercube used, as only the MPC protocol changed.

We firstly prove that an honest prover is always accepted. Conversely, a prover who commits a bad witness in the first step of the protocol has a probability lower than $\epsilon = \big(p + (1-p)\frac{1}{N}\big)$ of being accepted. Consequently, a prover that has a higher probability of acceptance knows the secret.

**Theorem 1.** *The protocol in Fig. 3 is perfectly complete. Namely, a prover who knows $x$ and performs the protocol correctly will be accepted by a verifier with probability 1.*

*Proof.* By construction, if the prover has knowledge of a solution of the Rank-SD instance, he will always be able to execute the protocol correctly.

**Theorem 2.** *If an efficient prover $\tilde{\mathcal{P}}$ with knowledge of only public data $(\boldsymbol{H}, \boldsymbol{y})$ can convince a verifier $\mathcal{V}$ with probability:*

$$\tilde{\epsilon} = \mathsf{Pr}\big(\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle \rightarrow ACCEPT\big) > \epsilon = p + (1-p)\frac{1}{N}$$

*then there exists an extraction function $\mathcal{E}$ that produces a commitment collision, or a good witness $\boldsymbol{x}'$ solution of the Rank-SD instance, by making an average number of calls to $\tilde{\mathcal{P}}$ upper bounded by:*

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

*Proof.* The proof is similar to that of [AMGH+22]. We first need to establish that the probability for the malicious prover (who has no knowledge of the solution of the Rank-SD instance used) to cheat is at most $\epsilon = \frac{1}{N} + (1-\frac{1}{N})\cdot p$.
There are two situations where a malicious prover can be accepted by the verifier:

- He obtains the value $v = 0$ when executing the MPC protocol thanks to a false positive.
- The verifier believes that the value obtained $v$ is 0.

The first case occurs with probability $p = \frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}$, as false positive rate of the protocol $\Pi^{\eta}$.

In the second case, the malicious prover needs to alter the communications in order to fool the verifier. Among the $N$ shares, only the share $i^*$ will not be revealed by the prover. This means that, if he cheats on more than one share, the verifier will notice the cheating, and thus rejects the proof. However, if he cheats on zero share, he will be rejected as well since the value $v$ will not be 0 since the malicious prover does not have a good witness.

This means the malicious prover can only cheat on one share exactly. However, cheating on one share $s$ means that this will show on one main share of all the $D$ dimensions, as the main shares are the sum of leaves that have the same index $i_k$ along the current dimension. This means that the cheating is not detected if and only if the share the prover cheated on is $i^*$, since there exists a bijection between leaves and the set of their associated main parties. This happens with probability $\frac{1}{N}$ (as the prover does not know the value of $i^*$ before cheating). No other cheating is possible, because all leaf parties except $i^*$ are revealed. Remark that cheating on the main parties give the same probability of success, since the prover should cheat for each dimension on the main party associated to the unrevealed leaf $i^*$.

Thus, the probability for the malicious prover to fool the verifier is at most: $p + (1-p)\cdot\frac{1}{N} = \frac{1}{N} + (1-\frac{1}{N})\cdot p$.

We then need to show that the soundness is exactly this value, which we will do in what follows:
let $\mathcal{T}_1$ and $\mathcal{T}_2$ two transcripts with the same commitments, i.e, the same $h_0 = \mathsf{H}(\mathsf{cmt}_1, \ldots, \mathsf{cmt}_N)$ , but the second challenges $i_1^*$ (for $\mathcal{T}_1$) and $i_2^*$ (for $\mathcal{T}_2$) differ.

Then, we have two possibilities:

- $[\![\boldsymbol{x}_B]\!], [\![\boldsymbol{\beta}]\!], [\![\boldsymbol{a}]\!]$ and $[\![c]\!]$ differ in the two transcripts, and the malicious prover found a collision in the commitment hash.
- the openings of the commitments are equal, and thus the shares $[\![\boldsymbol{x}_B]\!], [\![\boldsymbol{\beta}]\!], [\![\boldsymbol{a}]\!]$ and $[\![c]\!]$ are equal in the transcripts.

We will only consider the second case, as we suppose that we use secure hash functions and secure commitment schemes.

Then, since $i_1^*$ and $i_2^*$ are different challenges and the commitments are the same, it is possible to recover the witness. We will then show that this means we can build an extractor function in order to obtain a good witness.

$\boldsymbol{x}$ is called a good witness if it is solution to the Rank-SD instance defined by public data, i.e. $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{y}$ and $W_R(\boldsymbol{x}) \leq r$. Let $R_h$ the random variable associated to the randomness in initial commitment, and $r_h$ is the value it takes.

For a malicious prover $\tilde{\mathcal{P}}$ to be able to get these two transcripts, he does the following:

- Run the protocol with randomness $r_h$ with the verifier until $\mathcal{T}_1$ is found, i.e, $\mathcal{T}_1$ is the first accepted transcript found by $\tilde{\mathcal{P}}$. We note $i_1^*$ the leaf challenge obtained.

– Then, using the same randomness $r_h$ that was used, i.e, building the same commitments, $\tilde{\mathcal{P}}$ repeats the process, until he finds another accepted transcript, $\mathcal{T}_2$, for which the leaf challenge, $i_2^*$, is different than $i_1^*$.

– He then recovers the witness. If it is a bad witness, he repeats from the beginning.

In order to establish the soundness of the protocol, we need to estimate the number of times a malicious prover needs to repeat the authentication protocol in order to get the good witness $\boldsymbol{x}$.
Let $\delta \in ]0,1[$, and $\tilde{\epsilon}$ such that $(1-\delta)\cdot\tilde{\epsilon} > \epsilon$. We will define the randomness $r_h$ to be a good randomness if $\mathsf{Pr}(\mathrm{Succ}_{\tilde{\mathcal{P}}}\,|r_h) > (1-\delta)\cdot\tilde{\epsilon}$.

By the Splitting Lemma, we have that $\mathsf{Pr}(r_h\ good|\,\mathrm{Succ}_{\tilde{\mathcal{P}}}) \geq \delta$. This means that after $\frac{1}{\delta}$ accepted transcripts, we have good odds to obtain a good randomness. Furthermore, we know that if the malicious prover uses a bad witness, his probability to cheat is bounded from above by $\epsilon$. Since the probability of success is greater than $\epsilon$, this means that a good witness has been used (when $r_h$ is good).

To continue this proof, we will look at the probability to have, given an accepted transcript $\mathcal{T}_1$, a second accepted transcript, $\mathcal{T}_2$, with a challenge different than the one in $\mathcal{T}_1$. This means we are looking to bound from below the probability:

$$\mathsf{Pr}(\mathrm{Succ}_{\tilde{\mathcal{P}}} \cap (i_1^* \neq i_2^*)|r_h\ good)$$

Trivially, we know that this probability is equal to the probability of success knowing that $r_h$ is good, minus the probability of success with $i_1^* = i_2^*$ knowing $r_h$ is good. This means:

$$\begin{aligned}
\mathsf{Pr}(\mathrm{Succ}_{\tilde{\mathcal{P}}} \cap (i_1^* \neq i_2^*)|r_h\ good) &= \mathsf{Pr}(\mathrm{Succ}_{\tilde{\mathcal{P}}}\,|r_h\ good) - \mathsf{Pr}(\mathrm{Succ}_{\tilde{\mathcal{P}}} \cap (i_1^* = i_2^*)|r_h\ good) \\
&\geq \mathsf{Pr}(\mathrm{Succ}_{\tilde{\mathcal{P}}}\,|r_h\ good) - \frac{1}{N} \\
&\geq (1-\delta)\tilde{\epsilon} - \frac{1}{N} \\
&\geq (1-\delta)\tilde{\epsilon} - \epsilon \qquad (\text{since } \epsilon \geq \frac{1}{N} \text{ trivially})
\end{aligned}$$

Now that we have this lower bound, we want to estimate the number of time one has to repeat the protocol to find $\mathcal{T}_2$. For that, we take the opposite probability, i.e, $1 - \mathsf{Pr}(\mathrm{Success}_{\tilde{\mathcal{P}}} \cap (i_1^* \neq i_2^*)|r_h\ good)$, which is lower bounded by $1 - ((1-\delta)\tilde{\epsilon} - \epsilon)$. We now want a probability of $\frac{1}{2}$ at least of success after $L$ tries of the authentication protocol. This means then that we want:

$$\begin{aligned}
\Big(1 - \mathsf{Pr}(\mathrm{Success}_{\tilde{\mathcal{P}}} \cap (i_1^* \neq i_2^*)|r_h\ good)\Big)^L &< \frac{1}{2} \\
L \cdot \ln(1 - ((1-\delta)\tilde{\epsilon} - \epsilon)) &< -\ln(2) \\
L &> -\frac{\ln(2)}{\ln(1 - ((1-\delta)\tilde{\epsilon} - \epsilon))}
\end{aligned}$$

One obtains the following majoration for the number of calls to $\tilde{\mathcal{P}}$:

$$L > \frac{\ln(2)}{\ln(\frac{1}{1-((1-\delta)\tilde{\epsilon}-\epsilon)})} \approx \frac{\ln(2)}{(1-\delta)\tilde{\epsilon} - \epsilon}$$

This means that, when repeating the protocol $L$ times, the probability to get the second transcript is higher than $\frac{1}{2}$.

Finally, we can look at the number of protocol repetitions that have to be done. To quickly remind the steps of the extraction:

– $\tilde{\mathcal{P}}$ repeats the authentication protocol until he finds an accepted transcript $\mathcal{T}_1$, where the commitments are generated by $r_h$, and with second challenge $i_1^*$.

– When $\mathcal{T}_1$ is found, repeat the protocol with the same value $r_h$, $L$ times. After that, $\tilde{\mathcal{P}}$ has more than $\frac{1}{2}$ chance of being successful. If he is not, he repeats from the first step of the procedure.

We will note $\mathbb{E}(\tilde{\mathcal{P}})$ the number of repetitions $\tilde{\mathcal{P}}$ has to make. After $L$ calls (to find $\mathcal{T}_2$), if $r_h$ is good (which happens with probability $\delta$), we have $\frac{1}{2}$ chance of not finding $\mathcal{T}_2$. However, if $r_h$ is not good (with probability $1-\delta$), then we can consider that $\mathcal{T}_2$ is never found.

Thus, $\mathsf{Pr}(\text{no } \mathcal{T}_2 | \, \text{Succ}_{\tilde{\mathcal{P}}}) = \frac{\delta}{2} + (1-\delta) = 1 - \frac{\delta}{2}$. If that happens, then, $\tilde{\mathcal{P}}$ has to return to the first step, i.e, find $\mathcal{T}_1$ again. This means:

$$\mathbb{E}(\tilde{\mathcal{P}}) \leq 1 + \Big((1 - \mathsf{Pr}(\text{Succ}_{\tilde{\mathcal{P}}}))\mathbb{E}(\tilde{\mathcal{P}})\Big) + \mathsf{Pr}(\text{Succ}_{\tilde{\mathcal{P}}})\Big(L + (1 - \frac{\delta}{2})\mathbb{E}(\tilde{\mathcal{P}})\Big)$$

Obviously, $\tilde{\mathcal{P}}$ needs to run at least once. Then, we need to add to that the number of times expected before finding $\mathcal{T}_1$, and then, the number of times expected before finding $\mathcal{T}_2$.
Since $\mathsf{Pr}(\text{Succ}_{\tilde{\mathcal{P}}}) = \tilde{\epsilon}$ (by assumption), we can replace, and simplify the expression. We find then:

$$\begin{aligned}
\mathbb{E}(\tilde{\mathcal{P}}) &\leq 1 + \Big((1 - \tilde{\epsilon} \cdot \mathbb{E}(\tilde{\mathcal{P}})\Big) + \tilde{\epsilon} \cdot \Big(L + (1 - \frac{\delta}{2}) \cdot \mathbb{E}(\tilde{\mathcal{P}})\Big) \\
\mathbb{E}(\tilde{\mathcal{P}}) &\leq 1 + \mathbb{E}(\tilde{\mathcal{P}}) - \tilde{\epsilon} \cdot \mathbb{E}(\tilde{\mathcal{P}}) + \tilde{\epsilon} \cdot L + \tilde{\epsilon} \cdot \mathbb{E}(\tilde{\mathcal{P}}) - \tilde{\epsilon} \cdot \frac{\delta}{2} \cdot \mathbb{E}(\tilde{\mathcal{P}}) \\
\tilde{\epsilon} \cdot \frac{\delta}{2} \cdot \mathbb{E}(\tilde{\mathcal{P}}) &\leq 1 + \tilde{\epsilon} \cdot L \\
\mathbb{E}(\tilde{\mathcal{P}}) &\leq \frac{2}{\tilde{\epsilon} \cdot \delta}\Big(1 + \tilde{\epsilon} \cdot L\Big) = \frac{2}{\tilde{\epsilon} \cdot \delta}\Big(1 + \tilde{\epsilon} \cdot \frac{\ln(2)}{(1-\delta)\tilde{\epsilon} - \epsilon}\Big)
\end{aligned}$$

Since this equality holds for any $\delta \in ]0, 1[$, we can take $\delta$ such that $(1-\delta)\tilde{\epsilon} = \frac{1}{2}(\tilde{\epsilon} + \epsilon)$, and thus, we obtain the result:

$$\mathbb{E}(\tilde{\mathcal{P}}) \leq \frac{4}{\tilde{\epsilon} - \epsilon} \cdot \Big(1 + 2\tilde{\epsilon} \cdot \frac{\ln(2)}{\tilde{\epsilon} - \epsilon}\Big)$$

This means we found an upper bound on the number of iterations $\tilde{\mathcal{P}}$ has to do before being able to retrieve a good witness, in the case where the probability to cheat was higher than $\epsilon$. This proves that $\epsilon$ is exactly the value $\frac{1}{N} + (1 - \frac{1}{N}) \cdot p$.

We now have to prove zero-knowledge property by building a simulator which outputs indistinguishable transcripts with the distribution of transcripts from honest executions of the protocol.

The main intuition is that if the prover knows the challenge before committing the initial shares, then he knows for which party to cheat. We consequently build an HVZK simulator which firstly generate the challenge to build valid transcriptions of the protocol.

**Theorem 3.** *If the algorithm PRG is a secure pseudo-random generator and the commitment* $\mathsf{Com}$ *is hiding, then the algorithm 11 is Honest-Verifier Zero Knowledge.*

*Proof.* Consider a simulator, described in Fig. 11, which produces the transcript responses $(h_0, \mathsf{ch}_1, h_1, \mathsf{ch}_2, \mathsf{rsp})$. We demonstrate that this simulator produces indistinguishable transcripts from the real distribution (the one that we would obtain if it were generated by an honest prover who knows $\boldsymbol{x}$) by considering a succession of simulators: we begin by a simulator which produces true transcripts, and and change it gradually until arriving the following simulator. We explains why the distribution of transcripts is always the same at each step.

- **Simulator 0 (real world):** It correctly executes the algorithm 3, hence its output is the correct distribution.
- **Simulator 1:** Same as the **Simulator 0**, but starts by sampling the random challenges, and then uses true randomness instead of seed-derived randomness for leaf $i^*$. If $i^* = N$, the leafs $[\![\boldsymbol{x}_B]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N$ are computed as in the MPC protocol.

**Public data** : An instance of a Rank-SD problem : $\boldsymbol{H} = (I\ \boldsymbol{H}') \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $\boldsymbol{y} \in \mathbb{F}_{q^m}^{n-k}$

**Step 1: Sample challenges**

1. Sample challenges :

- First challenge : $\mathsf{ch}_1 = \big((\gamma_j)_{j\in[1,n]}, \epsilon\big) \xleftarrow{\$} \mathbb{F}_{q^{m\cdot\eta}}^{n} \times \mathbb{F}_{q^{m\cdot\eta}}$
- Second challenge : $\mathsf{ch}_2 = i^* \xleftarrow{\$} [1, N]$

**Step 2: Compute shares and their commitments**

2. Sample a seed for pseudo-random generator : $\mathsf{seed} \xleftarrow{\$} \{0,1\}^{\lambda}$
3. Expand root seed recursively using TreePRG to obtain $N$ leafs and seeds $(\mathsf{seed}_i, \rho_i)$.
4. For each $i \in [1, N]\backslash\{i^*\}$ :

- Sample $[\![\boldsymbol{a}]\!]_i \xleftarrow{\$,\mathsf{seed}_i} \mathbb{F}_{q^{m\cdot\eta}}^{r-1}$

If $i \neq N$

- Sample $([\![\boldsymbol{x}_B]\!]_i, [\![\boldsymbol{\beta}]\!]_i, [\![c]\!]_i) \xleftarrow{\$,\mathsf{seed}_i} TPRG$
- $\mathsf{state}_i = \mathsf{seed}_i$

- Else:

- Sample $([\![\boldsymbol{x}_B]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N) \xleftarrow{\$} \mathbb{F}_{q^m}^{k} \times \mathbb{F}_{q^m}^{r} \times \mathbb{F}_{q^{m\cdot\eta}}$
- $\mathsf{aux}_N = ([\![\boldsymbol{x}_B]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N)$
- $\mathsf{state}_N = \mathsf{seed}_N \,||\, \mathsf{aux}_N$

- Simulate the computation of the party $i$ to get $[\![\boldsymbol{\alpha}]\!]_i$ and $[\![v]\!]_i$.

5. For the party $i^*$:

- $[\![\boldsymbol{\alpha}]\!]_{i^*} \xleftarrow{\$} \mathbb{F}_{q^{m\cdot\eta}}^{r-1}$
- $[\![v]\!]_{i^*} = -\sum_{i\neq i^*} [\![v]\!]_i$

6. Compute the commitment: $\mathsf{cmt}_{i^*} = \mathsf{Com}\big(\mathsf{state}_{i^*}, \rho_{i^*}\big)$.
7. Commit the full hypercube: $h_0 = \mathsf{Com}\big(cmt_1, ..., cmt_N\big)$.

8. For each main party $p \in [1, 2]$ : compute $[\![\boldsymbol{\alpha}]\!]_p$ and $[\![v]\!]_p$
9. For each dimension $k \in [1, D]$ : compute $H_k = \mathsf{H}([\![\boldsymbol{\alpha}]\!], [\![v]\!])$
10. Compute $h_1 = \mathsf{H}(H_1, ..., H_D)$

**Step 3 : Output transcript**

11. The prover outputs the transcript $(h_0, \mathsf{ch}_1, \mathsf{rsp}_1, \mathsf{ch}_2, \mathsf{rsp}_2)$, where $\mathsf{rsp}_1 = h_1$ and $\mathsf{rsp}_2 = \big(\mathsf{cmt}_{i^*}, [\![\boldsymbol{\alpha}]\!]_{i^*}, (\mathsf{state}_j, \rho_j)_{j\neq i^*}\big)$.

Fig. 11: HVZK simulator for zero-knowledge proof optimized with the hypercube technique

The pseudo-random generator is supposed to be $(t, \epsilon_{PRG})$-secure, its outputs are indistinguishable from the uniform distribution. Since the principal parts correspond to the sum of a certain number of leaves whose distributions are indistinguishable from that of the real world, their distribution is also indistinguishable.

– **Simulator 2:** Replace the leafs $[\![\boldsymbol{x}_B]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N$ in **Simulator 1** by uniform sampled values. Compute $[\![v]\!]_{i^*} = -\sum_{i\neq i^*} [\![v]\!]_i$. Note that this simulator becomes independent of the secret witness $\boldsymbol{x}_B$.

If $i^* = N$, it only impacts the shares $[\![\boldsymbol{\alpha}]\!]_{i^*}$ and $[\![v]\!]_{i^*}$. Note that this change does not alter the uniform distribution of these values. It does not alter the distribution of any other leaf.

If $i^* \neq N$, it only impacts $\mathsf{aux}_N = ([\![\boldsymbol{x}_B]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N)$ in the simulated response and the values computed from them in the MPC protocol. It does not alters the distribution of other leaves. We observe that the shares in $\mathsf{aux}_N$ are calculated by adding a randomness value from each seed of party $i \neq i^*$, which amounts to adding a uniform random value from $\mathsf{seed}_{i^*}$. Since this distribution was uniform too in **Simulator 1**, the output distributions are the same. Remember that this does not change the distributions of the main parts for the same reasons as before.

– **Simulator 3:** Rather than computing the value of $[\![\boldsymbol{\alpha}]\!]_{i^*}$ as in the MPC protocol, Simulator 3 samples it uniformly at random from $\mathbb{F}_{q^{m\cdot\eta}}^{r-1}$. As in the previous simulator, it does not change their output distribution.

As such, the output of the simulator is indistinguishable from the real distribution.

**5.1.2 MPC with threshold** We discuss here the security properties of the threshold protocol. The proofs will only be sketched, as they are very similar to those of the hypercube protocol.

**Theorem 4.** *The protocol in Fig. 8 is perfectly complete. Namely, a prover who knows $x$ and performs the protocol correctly will be accepted by a verifier with probability 1.*

*Proof.* Same trivial justification as in the previous case of the hypercube.

**Theorem 5.** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that on input $(\boldsymbol{H}, \boldsymbol{y})$ convinces an honest verifier $\mathcal{V}$ to accept with probability*

$$\tilde{\epsilon} = \mathsf{Pr}\big(\langle\tilde{\mathcal{P}}, \mathcal{V}\rangle \to ACCEPT\big) > \epsilon = \frac{1}{\binom{\ell}{N}} + p \cdot \frac{\ell \cdot (N-\ell)}{\ell+1}$$

*then there exists an extraction algorithm $\mathcal{E}$ that produces a commitment collision, or a good witness $\boldsymbol{x'}$ solution of the Rank-SD instance, by making an average number of calls to $\tilde{\mathcal{P}}$ bounded from above:*

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

*Proof.* For this proof, we refer to [FR22], which proves this theorem for any MPC protocol fitting this model. Our threshold protocol is an exact application of this model. Hence, the proof of the above theorem is the same as the proof in appendix D of [FR22].

**Theorem 6.** *If the algorithm PRG is pseudo-random generator and the commitment* $\mathsf{Com}$ *is hiding, then there exists an honest-Verifier Zero Knowledge algorithm.*

The proof is similar to what is done in the case of the additive sharing, and holds by the $t$-privacy of the MPC protocol, as well as the hiding property of the commitments. Moreover, a proof in the general case is provided in [FR22, Appendix C]. Since we are in their model of MPCitH, the proof applies here as well.

### 5.2 Security proof for signature schemes

As in the previous subsection, the proofs are largely inspired by the work of [AMGH+22], which is largely inspired by [FJR22]. The two proofs are very similar, and proceed in two main steps: we begin by explaining how efficiently simulate a signature oracle to a CMA adversary using public key and the honest verifier zero-knowledge simulator. We then show how such a RUF-CMA adversary can be used to extract a solution to the Rank-SD instance.

**5.2.1 Additive-RYDE** We begin by computing the probability of an attack on the signature based in the hypercube MPC.

**Theorem 7.** *Let the signature be $(t, \epsilon_{PRG})$-secure and with adversary of advantage at most $\epsilon_{RSD}$ against the underlying Rank-SD problem. Let $\mathsf{H}_0$, $\mathsf{H}_1$, $\mathsf{H}_2$, $\mathsf{H}_3$, and $\mathsf{H}_4$ be random oracles with output length $2\lambda$ bits. If an adversary makes $q_i$ queries to $\mathsf{H}_i$ and $q_S$ queries to the signature oracle, the probability of such an adversary producing an existential forgery under chosen message attack (EUF-CMA) is bounded from above by:*

$$\mathsf{Pr}[\mathsf{Forge}] \;\leq\; \frac{3 \cdot (q + \tau \cdot N \cdot q_S)^2}{2 \cdot 2^\lambda} + \frac{q_S \cdot (q_S + 5q)}{2^\lambda}] + \epsilon_{PRG} + \mathsf{Pr}[X + Y = \tau] + \epsilon_{RSD}$$

*where* $q = \max\{q_0, q_1, q_2, q_3, q_4\}$*,* $\tau$ *is the number of rounds of the signature,* $p = \frac{1}{q^{m\eta}} + \left(1 - \frac{1}{q^{m\eta}}\right)\frac{1}{q^{m\eta}}$ *is the false positive rate of the underlying MPC protocol,* $X = \max_{i\in[0,q_2]}\{X_i\}$ *with* $X_i \sim \mathcal{B}(\tau, p)$*, and* $Y = \max_{i\in[0,q_4]}\{Y_i\}$ *with* $Y_i \sim \mathcal{B}(\tau - X, \frac{1}{N})$*.*

*Proof.* In this proof, we will adopt a game hopping strategy in order to find the upper bound. We note $\mathsf{Pr}_i[\mathsf{Forge}]$ the probability of forgery when considering game $i$.

The first game will be the access to the standard signing oracle by the adversary $\mathcal{A}$. We will then game hop in order to eliminate the cases where collisions happen, and, through some other games, we will manage to find an upper bound. The aim of the proof is to find this bound on $\mathsf{Pr}_1[\mathsf{Forge}]$.

– **Game 1:**
This is the interaction between $\mathcal{A}$ and the real signature scheme.
$\mathsf{KeyGen}$ generates $(\boldsymbol{H}, \boldsymbol{y})$ and $\boldsymbol{x}$, and $\mathcal{A}$ receives $(\boldsymbol{H}, \boldsymbol{y})$. $\mathcal{A}$ can make queries to each $\mathsf{H}_i$ independently, and can make signing queries. At the end of the attack, $\mathcal{A}$ outputs a message/signature pair, $(m, \sigma)$. The event $\mathsf{Forge}$ happens when the message output by $\mathcal{A}$ was not previously used in a query to the signing oracle.

– **Game 2:**
In this game, we add a condition to the success of the attacker. The condition we add is that if there is a collision between outputs of $\mathsf{H}_0$, or $\mathsf{H}_1$, or $\mathsf{H}_3$, then, the forgery isn't valid.
The first step is to look at the number of times every $\mathsf{H}_i$ is called when calling the signing oracle. For $\mathsf{H}_0$, we make $\tau \cdot N$ queries. The signing oracle contains also $\tau$ calls to $\mathsf{H}_1$, one to $\mathsf{H}_2$, $\tau \cdot D$ to $\mathsf{H}_3$, and finally, a single one to $\mathsf{H}_4$.
The number of queries to $\mathsf{H}_0$ or $\mathsf{H}_1$ or $\mathsf{H}_3$ is then bounded from above by $q + \tau \cdot N \cdot q_S$.
We can then have the following result (it comes simply from the probability to have at least one collision with $q + \tau \cdot N \cdot q_S$ values):

$$|\mathsf{Pr}_1[\mathsf{Forge}] - \mathsf{Pr}_2[\mathsf{Forge}]| \leq \frac{3 \cdot (q + \tau \cdot N \cdot q_S)^2}{2 \cdot 2^\lambda}$$

– **Game 3:**
The attacker now fails if the inputs to any of the $\mathsf{H}_i$ has already appeared in a previous query. If that happens, this means that at least the salt used was the same (we emphasize on *at least*). We have one $\mathsf{salt}$ sampled every time a query is made to the signing oracle, and it can collide each time with: a previous $\mathsf{salt}$, or any of the queries to the $\mathsf{H}_i$. This means, we can bound this with:

$$|\mathsf{Pr}_2[\mathsf{Forge}] - \mathsf{Pr}_3[\mathsf{Forge}]| \leq \frac{q_S \cdot (q_S + q_0 + q_1 + q_2 + q_3 + q_4)}{2^\lambda} \leq \frac{q_S \cdot (q_S + 5q)}{2^\lambda}$$

– **Game 4:**
To answer the signing queries, we now use the **HVZK** simulator built in the previous proof, in order to generate the views of the open parties. By security of the PRG, the difference with the previous game is:

$$|\mathsf{Pr}_7[\mathsf{Forge}] - \mathsf{Pr}_6[\mathsf{Forge}]| \leq \epsilon_{PRG}$$

– **Game 5:**
Finally, we say that an execution $e^*$ of a query $h_2 = \mathsf{H}_4(m, \mathsf{salt}, h_1, H_1^{(e)} \dots H_D^{(e)}{}_{e\in[1,\tau]})$ defines a good witness $\boldsymbol{x}$ if:

- Each of the $H_k^{(e)}$ are the output of a query to $\mathsf{H}_3$
- $h_1$ is the output of a query to $\mathsf{H}_2$, i.e.:

$$h_1 = \mathsf{H}_2(\mathsf{salt}, m, h_0^{(1)}, \dots, h_0^{(\tau)})$$

- Each $h_0^{(e)}$ is the output of a query to $\mathsf{H}_1$, i.e.:

$$h_0^{(e)} = \mathsf{H}_1(\mathsf{cmt}_1^{(e)}, \ldots, \mathsf{cmt}_N^{(e)})$$

- Each $\mathsf{cmt}_i^{(e)}$ is the output of a query to $\mathsf{H}_0$, i.e.:

$$\mathsf{cmt}_i^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{(e)})$$

- The vector $\boldsymbol{x} \in \mathbb{F}_q^k$ defined by states $\{\mathsf{state}_i\}_{i\in[1,N]}$ is a correct witness, i.e.: $\boldsymbol{Hx} = \boldsymbol{y}$ such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$. In the case where an execution like this happens, we are able to retrieve the correct witness from the states $\{\mathsf{state}_i\}_{i\in[1,N]}$, and as a consequence, we are able to solve the Rank-SD instance. This means that $\mathsf{Pr}_8[\mathsf{Solve}] \leq \epsilon_{RSD}$.

Finally, we only need to look at the upper bound of $\left|\mathsf{Pr}_8[\mathsf{Forge} \cap \overline{\mathsf{Solve}}]\right|$. This probability is upper bounded by the value

$$\mathsf{Pr}[X + Y = \tau]$$

where $X = \max_{i\in[0,q_2]}\{X_i\}$ with $X_i \sim \mathcal{B}(\tau, p)$, $Y = \max_{i\in[0,q_4]}\{Y_i\}$ with $Y_i \sim \mathcal{B}(\tau - X, \frac{1}{N})$. We explain this bound below:

$\mathsf{Solve}$ does not happen here, meaning that, to have a forgery after a query to $\mathsf{H}_4$, $\mathcal{A}$ has no choice but to cheat either on the first round or on the second one.

*Cheating at the first round.* For any query $Q_2$ to $\mathsf{H}_2$, we call the output of this query $h_1$. For any query $Q_2$, if a false positive appears in a round $e$ with this value of $h_1$, then we add this round $e$ to the set we call $G_2(Q_2, h_1)$. This means that $\mathsf{Pr}(e \in G_2(Q_2, h_1) \mid \overline{\mathsf{Solve}}) \leq p$. Since the response $h_1$ is uniformly sampled, each round $e$ has the same probability to be in the set $G_2(Q_2, h_1)$. This means that $\#G_2(Q_2, h_1)$ follows the binomial distribution $X_{Q_2} = \mathcal{B}(\tau, p)$. We can then define $(Q_{2\text{best}}, h_{1\text{best}})$ such that $\#G_2(Q_2, h_1)$ is maximized, i.e,

$$\#G_2(Q_{2\text{best}}, h_{1\text{best}}) \sim X = \max\{X_{Q_2}\}_{(Q_2\in\mathcal{Q}_2)}$$

*Cheating at the second round.* Now, we need to look at the cheating in the second round, i.e, the queries to $\mathsf{H}_4$. We will note this query $Q_4$, with the output of this query $h_2$. For the signature to be accepted, we know that, if in a round, the prover sends a wrong value of $h_1$, then he needs to cheat on exactly one party (it is already established that is isn't possible to cheat on less, or on more, than one party). He only needs to cheat when the value of $h_1^{(e)}$ is wrong, i.e, he needs to cheat for every round $e \notin G_2(Q_{2\text{best}}, h_{1\text{best}})$. Since every time he cheats, the probability to be detected is $\frac{1}{N}$, it is easy to see the probability that the verification outputs ACCEPT is upper bounded by $\left(\frac{1}{N}\right)^{\tau - \#G_2(Q_{2\text{best}}, h_{1\text{best}})}$

The probability that the prover is accepted on one of the $q_4$ queries is then upper bounded by $1 - \left(1 - \left(\frac{1}{N}\right)^{\tau-\tau_1}\right)$ where $\tau_1 = \#G_2(Q_{2\text{best}}, h_{1\text{best}})$. By summing over all values of $\tau_1$ possible, we have then the upper bound:

$$\mathsf{Pr}_8(\mathsf{Forge} \cap \overline{\mathsf{Solve}}) \leq \mathsf{Pr}(X + Y = \tau)$$

where $X$ is as before, and $Y = \max\{Y_{Q_2}\}_{(Q_2\in\mathcal{Q}_2)}$ where the $Y_{Q_2}$ are distributed following $\mathcal{B}(\tau - X, \frac{1}{N})$.

All that is left to do is then to compute the sum of all the upper bounds we retrieved: this gives us the wanted result.

**5.2.2 Threshold-RYDE** We compute now the probability to forge the signature based on the MPC threshold.

**Theorem 8.** *Let the signature be $(t, \epsilon_{PRG})$-secure and with adversary of advantage at most $\epsilon_{RSD}$ against the underlying Rank-SD problem. Let $\mathsf{H}_0$, $\mathsf{H}_1$, $\mathsf{H}_2$ and $\mathsf{H}_M$ be random oracles with output length $2\lambda$ bits. The probability of such an adversary producing an existential forgery under chosen message attack (EUF-CMA) is bounded from above by:*

$$\mathsf{Pr}[\mathsf{Forge}] \leq \frac{\cdot(q + \tau \cdot (2N-1) \cdot q_S)^2}{2^\lambda} + \frac{q_S \cdot (q_S + 3q)}{2^\lambda} + q_S \cdot \tau \cdot \epsilon_{PRG} + \mathsf{Pr}[X + Y = \tau] + \epsilon_{RSD}$$

*where $\tau$ is the number of rounds of the signature, $p = \frac{1}{q^{m\eta}} + \left(1 - \frac{1}{q^{m\eta}}\right) \frac{1}{q^{m\eta}}$ the false positive rate of the MPC protocol, $X = \max_{i \in [0, q_1]}\{X_i\}$ with $X_i \sim \mathcal{B}(\tau, \binom{N}{\ell+1} \cdot p)$, and $Y = \max_{i \in [0, q_2]}\{Y_i\}$ with $Y_i \sim \mathcal{B}\left(\tau - X, \frac{1}{\binom{N}{\ell}}\right)$.*

*Proof.* The proof is based on the same operation as the previous one, adopting a game hopping strategy. The first game will be the access to the standard signing oracle by the adversary $\mathcal{A}$. The aim of the proof is to find this bound on $\mathsf{Pr}_1[\mathsf{Forge}]$.

– **Game 1:**
This is the interaction between $\mathcal{A}$ and the real signature scheme.
$\mathsf{KeyGen}$ generates $(\boldsymbol{H}, \boldsymbol{y})$ and $\boldsymbol{x}$, and $\mathcal{A}$ receives $(\boldsymbol{H}, \boldsymbol{y})$. $\mathcal{A}$ can make queries to each $\mathsf{H}_i$ independently, and can make signing queries. At the end of the attack, $\mathcal{A}$ outputs a message/signature pair, $(m, \sigma)$. The event $\mathsf{Forge}$ happens when the message output by $\mathcal{A}$ was not previously used in a query to the signing oracle.

– **Game 2:**
We add a condition to the success of the attacker now. If there is a collision in the outputs of $\mathsf{H}_0$ or on $\mathsf{H}_M$, then the forgery isn't valid. Here, $\mathsf{H}_0$ is called $q_0$ times by $\mathcal{A}$, $\mathsf{H}_M$ $q_M$ times. When $\mathcal{A}$ calls the signing oracle, there are in total: $\tau \cdot N$ calls to $\mathsf{H}_0$, $\tau \cdot (2N-1)$ calls to $\mathsf{H}_M$, and one to $\mathsf{H}_1$ and $\mathsf{H}_2$. In this game, only $\mathsf{H}_0$ and $\mathsf{H}_M$ are of interest. We can then give an upper bound to the queries made by $\mathcal{A}$ to the hash functions, which is then: $q + \tau \cdot (2N-1) \cdot q_S$ where $q_S$ is the number of queries to the signing oracle.
When making this many queries, we can now bound from above the probability of having a collision, with

$$|\mathsf{Pr}_1[\mathsf{Forge}] - \mathsf{Pr}_2[\mathsf{Forge}]| \leq \frac{(q + \tau \cdot (2N-1) \cdot q_S)^2}{2^\lambda}$$

– **Game 3:**
The attacker now fails if the inputs to any of the hash functions has already appeared in a previous query. If that happens, this means that at least the $\mathsf{salt}$ used was the same. Since we don't use $\mathsf{salt}$ in the Merkle Tree, this will only concern $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$. We sample $\mathsf{salt}$ $q_S$ time (once by signing oracle query), and $3q$ times as well (each time we call $\mathsf{H}_0, \mathsf{H}_1$ or $\mathsf{H}_2$). If there is an input which already appears for $\mathsf{H}_M$, this must be because a collision has been found either on $\mathsf{H}_M$ or on $\mathsf{H}_0$. However, we already excluded this in **Game 2**. This means we can give the following bound:

$$|\mathsf{Pr}_2[\mathsf{Forge}] - \mathsf{Pr}_3[\mathsf{Forge}]| \leq \frac{q_S \cdot (q_S + q_0 + q_1 + q_2)}{2^\lambda} \leq \frac{q_S \cdot (q_S + 3q)}{2^\lambda}$$

– **Game 4:**
To answer the signing queries, we now use the **HVZK** simulator built in the previous proof, in order to generate the views of the open parties. By security of the PRG, the difference with the previous game is:

$$|\mathsf{Pr}_7[\mathsf{Forge}] - \mathsf{Pr}_6[\mathsf{Forge}]| \leq \epsilon_{PRG}$$

– **Game 5:**
Finally, we say that an execution $e^*$ of a query $h_2 = \mathsf{H}_2(m, \mathsf{pk}, \mathsf{salt}, h_1, (\llbracket \boldsymbol{\alpha}^{(e)} \rrbracket_i, \llbracket v^{(e)} \rrbracket_i)_{i \in S, e \in [1,\tau]})$ defines a good witness $\boldsymbol{x}$ if:

- $h_1$ is the output of a query to $H_1$, i.e,

$$h_1 = \mathsf{H}_2(\mathsf{salt}, m, h_0^{(1)}, \ldots, h_0^{(\tau)})$$

- Each $h_0^{(e)}$ is the output of a query to the MerkleTree oracle, i.e,

$$h_0^{(e)} = \mathsf{Merkle}(\mathsf{cmt}_1^{(e)}, \ldots, \mathsf{cmt}_N^{(e)})$$

- Each $\mathsf{cmt}_i^{(e)}$ is the output of a query to $\mathsf{H}_0$, i.e,

$$\mathsf{cmt}_i^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{(e)})$$

- The vector $\boldsymbol{x} \in \mathbb{F}_q^k$ defined by states $\{\mathsf{state}_i\}_{i \in S}$ is a correct witness, i.e.: $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{y}$ such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$.

In the case where an execution like this happens, we are able to retrieve the correct witness from the states $\{\mathsf{state}_i\}_{i \in [1,N]}$, and as a consequence, we are able to solve the Rank-SD instance. This means that $\mathsf{Pr}_8[\mathsf{Solve}] \leq \epsilon_{RSD}$.

Finally, we only need to look at the upper bound of $\left|\mathsf{Pr}_8[\mathsf{Forge} \cap \overline{\mathsf{Solve}}]\right|$. This probability is upper bounded by the value

$$\mathsf{Pr}[X + Y = \tau]$$

with $X = \max_{i \in [0,q_2]}\{X_i\}$ with $X_i \sim \mathcal{B}(\tau, \binom{N}{\ell+1} \cdot p)$, $Y = \max_{i \in [0,q_4]}\{Y_i\}$ with $Y_i \sim \mathcal{B}(\tau - X, \frac{1}{\binom{N}{\ell}})$ and where $p = \frac{1}{q^{m\eta}} + \left(1 - \frac{1}{q^{m\eta}}\right) \frac{1}{q^{m\eta}}$.
This result comes directly from [FR22, Lemma 6 and Theorem 4, Appendix F].

All that is left to do is then to compute the sum of all the upper bounds we retrieved: this gives us the wanted result.

# 6 Known Attacks

## 6.1 Attacks against Fiat-Shamir signatures

There are several attacks against signatures from zero-knowledge proofs obtained thanks to the Fiat-Shamir heuristic. [AABN02] propose an attack more efficient than the brute-force one for protocols with more than one challenge, i.e. for protocols of a minimum of 5 rounds.

Kales and Zaverucha proposed in [KZ20] a forgery achieved by guessing separately the two challenges of the protocol. It results an additive cost rather than the expected multiplicative cost. The cost associated with forging a transcript that passes the first 5 rounds of the Proof of Knowledge (Fig.3 or Fig.8) relies on achieving an optimal trade-off between the work needed for passing the first step and the work needed for passing the second step. To achieve the attack, one can find an optimal number of repetitions with the formula:

$$\tau_1 = \arg \min_{0 \leq \tau \leq r} \left\{ \frac{1}{P_1} + P_2^{r-\tau_1} \right\}$$

where $P_1$ and $P_2$ are the probabilities to pass respectively the first and the second challenge.

In the case of the additive case, one obtains:

$$\mathrm{cost}_{\mathrm{forge}} = \min_{0 \leq \tau' \leq \tau} \left\{ \frac{1}{\sum_{i=\tau'}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i}} + (N)^{\tau-\tau'} \right\}$$

with $p = \frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}$.

In the case of low-threshold case, one obtains:

$$\text{cost}_{\text{forge}} = \min_{0 \leq \tau' \leq \tau} \left\{ \frac{1}{\sum_{i=\tau'}^{\tau} \binom{\tau}{i} p'^i (1-p')^{\tau - i}} + \binom{N}{\ell}^{\tau - \tau'} \right\}$$

with $p' = \left( \frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}} \right) \binom{N}{\ell+1}$.

### 6.2 Attacks against Rank-SD problem

We give the complexity of the main attacks against the Rank-Syndrome-Decoding problem. The interested reader can found the detailed attacks and proofs in [ABB+23].

The enumeration of basis is a combinatorial attack which consists in trying all the different possible supports fof the error. The complexity of this attack is upper bounded by:

$$O((nr+m)^3 q^{(m-r)(r-1)})$$

.

An other combinatorial attack is the error support attack, which is the adaptation of the information set decoding attack used in Hamming metric. It consists in guessing a set of $n-k$ coordinates which contains the support of the vector $\boldsymbol{x}$ to obtain a system of $n-k$ equations with $n-k$ variables, for which there exists often a solution. This attack can be achieved with an average complexity:

$$O((n-k)^3 m^3 q^{(r-1)\lfloor \frac{(k+1)m}{n} \rfloor})$$

There exists also algebraic attacks to solve the system of equations $\boldsymbol{Hx} = \boldsymbol{y}$ using computer algebra techniques like Gröbner basis. Today, the best algebraic modelings for solving the Rank-SD problem are the MaxMinors modeling [BBB+20,BBC+20] and the Support Minors modeling [BBC+20,BBB+22]. The final cost in $\mathbb{F}_q$ operations is given by:

$$\mathcal{O}\left(m^2 N M^{\omega-1}\right),$$

where

$$\begin{aligned} N = &\sum_{i=1}^{k} \binom{n-i}{r} \binom{k+b-1-i}{b-1} - \binom{n-k-1}{r} \binom{k+b-1}{b} \\ &- (m-1) \sum_{i=1}^{b} (-1)^{i+1} \binom{k+b-i-1}{b-i} \binom{n-k-1}{r+i}. \\ M = &\binom{k+b-1}{b} \left( \binom{n}{r} - m \binom{n-k-1}{r} \right). \end{aligned}$$

and $\omega$ is the linear algebra constant.

## 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, *Advances in Cryptology — EUROCRYPT 2002*, pages 418–433, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[ABB+23] Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Antoine Joux, Jean-Pierre Tillich, Matthieu Rivain, and Adrien Vinçotte. RYDE specifications. 2023.

[AFK21] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-Shamir Transformation of Multi-Round Interactive Proofs. Cryptology ePrint Archive, Paper 2021/1377, 2021. https://eprint.iacr.org/2021/1377.

[AMAB+20] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Maxime Bros, Alain Couvreur, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. Rank Quasi-Cyclic (RQC). *NIST Post-Quantum Cryptography Standardization Project (Round 2), https://pqc-rqc.org*, 2020.

[AMGH+22] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The Return of the SDitH. *Cryptology ePrint Archive, Report 2022/1645*, 2022.

[BBB+20] Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, Vincent Neiger, Olivier Ruatta, and Jean-Pierre Tillich. An algebraic attack on rank metric code-based cryptosystems. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 64–93, Cham, 2020. Springer International Publishing.

[BBB+22] Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, and Jean-Pierre Tillich. Revisiting algebraic attacks on minrank and on the rank decoding problem, 2022.

[BBC+20] Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier Verbel. Improvements of algebraic attacks for solving the rank decoding and minrank problems. In *Advances in Cryptology - ASIACRYPT 2020, International Conference on the Theory and Application of Cryptology and Information Security, 2020. Proceedings*, pages 507–536, 2020.

[BGRV23] Loïc Bidoux, Philippe Gaborit, Olivier Ruatta, and Adrien Vinçotte. MPCitH-based Signature for the RSD problem using a Hypercube. In *IEEE International Symposium on Information Theory (ISIT)*, 2023.

[BN20] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography – PKC 2020*, pages 495–526, Cham, 2020. Springer International Publishing.

[Fen22] Thibauld Feneuil. Building MPCitH-based Signatures from MQ, MinRank, Rank SD and PKP. *Cryptology ePrint Archive, Report 2022/1512*, 2022.

[FJR22] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome Decoding in the Head : Shorter Signatures from Zero-Knowledge Proofs. In *Annual International Cryptology Conference (CRYPTO)*, 2022.

[FR22] Thibauld Feneuil and Matthieu Rivain. Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head. Cryptology ePrint Archive, Paper 2022/1407, 2022.

[FS86] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *International Cryptology Conference (CRYPTO)*, 1986.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, aug 1986.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-Knowledge from Secure Multiparty Computation. In *Proceedings of the 39th annual ACM symposium on Theory of computing (STOC)*, 2007.

[KZ20] Daniel Kales and Greg Zaverucha. An Attack on Some Signature Schemes Constructed From Five-Pass Identification Schemes. In *International Conference on Cryptology and Network Security (CANS)*, 2020.

[Ore33] Oystein Ore. On a special class of polynomials. *Transactions of the American Mathematical Society*, 35(3):559–584, 1933.

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