MIRA: a Digital Signature Scheme based on the MinRank problem and the MPC-in-the-Head paradigm

Nicolas Aragon[1], Loïc Bidoux[2], Jesús-Javier Chi-Domínguez[2], Thibauld Feneuil[3,4], Philippe Gaborit[5], Romaric Neveu[5], and Matthieu Rivain[4]

[1] Naquidis Center, Talence, France
[2] Technology Innovation Institute, UAE
[3] Sorbonne Université, CNRS, INRIA, Institut de Mathématiques de Jussieu-Paris Rive Gauche, Ouragan, Paris, France
[4] CryptoExperts, Paris, France
[5] University of Limoges, France

Abstract. We exploit the idea of [Fen22] which proposes to build an efficient signature scheme based on a zero-knowledge proof of knowledge of a solution of a MinRank instance. The scheme uses the MPCitH paradigm, which is an efficient way to build ZK proofs. We combine this idea with another idea, the hypercube technique introduced in [AMGH⁺22], which leads to more efficient MPCitH-based scheme. This new approach is more efficient than classical MPCitH, as it allows to reduce the number of party computation. This gives us a first scheme called MIRA-Additive. We then present an other scheme, based on low-threshold secret sharings, called MIRA-Threshold, which is a faster scheme, at the price of larger signatures. The construction of MPCitH using threshold secret sharing is detailed in [FR22]. These two constructions allows us to be faster than classical MPCitH, with a size of signature around 5.6kB with MIRA-Additive, and 8.3kB with MIRA-Threshold. We detail here the constructions and optimizations of the schemes, as well as their security proofs.

Table of Contents

1 Introduction

MIRA is a signature scheme designed to be secure against attacks from a quantum computer. The scheme is based on the MPC-in-the-Head paradigm and its security relies on the hardness to solve the MinRank problem. The underlying proof of knowledge uses symmetric functions as ingredients, such as hash functions and commitment schemes.

In section 2, we remind the reader some notations and definitions. We remind as well the MPC protocol we will use. In section 3, we describe the two variants of the MPCitH scheme, one using additive secret sharing, the other using threshold secret sharing. Then, section 4 explains the choice of parameters, and the obtained theoretical sizes. Section 5 deals with the security proofs of both schemes. Finally, section 6 is dedicated to the security of the scheme regarding to the Fiat-Shamir transform and the MinRank problem.

In our schemes, the MPC protocol we use is the linearized-polynomial protocol on MinRank described in [Fen22]. The hypercube MPCitH idea comes from [AMGH+22], while the threshold MPCitH one from [FR22].

2 Preliminaries

2.1 Notations and Conventions

Let A a randomized algorithm. We write $y \leftarrow A(x)$ the output of the algorithm for the output 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 the set of integers between 1 and N by $[1, N]$, which can be shortened in $[N]$.

We denote by $\mathbb{F}_q$ the finite field of order q. We use bold letters to denote vectors or 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 λ is denoted $negl(\lambda)$ while a polynomial function in λ is denoted $poly(\lambda)$. We further use the notation $poly(\lambda_1, \lambda_2, ...)$ for a polynomial function in several variables.

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

$$\Pr[\mathcal{A}^{D_\lambda}() = 1] - \Pr[\mathcal{A}^{E_\lambda}() = 1] \leq \epsilon(\lambda)$$

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 $\epsilon \in \mathsf{negl}(\lambda)$ for every $t \in \mathsf{poly}(\lambda)$;

- statistically indistinguishable if $\epsilon \in \mathsf{negl}(\lambda)$ for every unbounded t;

- perfectly indistinguishable if $\epsilon = 0$ for every unbounded t.

2.1.1 Pseudorandom Generators

Definition 1 (Pseudorandom Generators). *Let $G : \{0,1\}^* \rightarrow \{0,1\}^*$, ℓ a polynomial such that $G(s) \in \{0,1\}^{\ell(\lambda)}$. G is a (t,ϵ)-secure pseudorandom generator if:*

- *$\ell(\lambda) > \lambda$,*

- *the distributions $\{G(s), s \leftarrow \{0,1\}^\lambda\}$ and $\{r, r \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}\}$ are indistinguishable.*

In some protocols, we are going to use TreePRG, which is a pseudorandom generator, which uses a root seed to generate N other seeds in a structured way. This can be illustrated quite easily with the following figure:

$$seed_{\text{root}}$$
$$seed_{int_1} \qquad seed_{int_2}$$
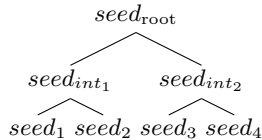$$seed_1 \; seed_2 \; seed_3 \; seed_4$$

Fig. 1: Example of TreePRG

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 when, 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 $N-1$. In general, for a TreePRG with N final seeds 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 $|I| \log_2(\frac{N}{|I|})$ tree nodes.

We thus have three functions in order to deal with a TreePRG:

- the root seed expansion, which generates N seeds from a root seed;

- the sibling path derivation, which generates a sibling path from a leaf seed;

- the leaf seeds retrieval from a sibling path.

2.1.2 Collision-Resistant Hash Functions
We are going to use hash functions in the zero-knowledge protocols. We define below what is a collision resistant hash function.

Definition 2 (Collision-Resistant Hash Function). *Let $h : \{0,1\}^* \rightarrow \{0,1\}^\lambda$. It is a collision-resistant hash function if h can be computed in a polynomial time, and, for any polynomial algorithm $\mathcal{A}$,*

$$\Pr[(x_1, x_2) \leftarrow \mathcal{A}(\lambda), x_1 \neq x_2, h(x_1) = h(x_2)] < \epsilon(\lambda)$$

where $\epsilon(\lambda)$ is negligeable.

2.1.3 Commitments Schemes

The security of the proof of knowledge relies on commitments, in order to avoid an attacker to forge a valid transcript. The commitment scheme should satisfy two properties: the commitment should reveal no information about the data we committed (hiding property), and there should be only one way to open the commitment (binding property).

Definition 3 (Commitment Scheme). *A commitment scheme is defined by the function* Com, *which takes as inputs* m, ρ *with* m, *and* ρ, *and outputs* c, *for some message space, randomness space, and commitment space.*

Definition 4 (Hiding). *A commitment scheme Com is computationally (resp. statistically, resp. perfectly) hiding if, for every* m_0, m_1, *the distributions of*

$$\{\mathsf{Com}(m_0, \rho), \rho \leftarrow\!\$\} \text{ and } \{\mathsf{Com}(m_1, \rho), \rho \leftarrow\!\$\}$$

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

Definition 5 (Binding). *A commitment scheme Com is binding if, for every PPT algorithm* $\mathcal{A}$, *we have*

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

where μ *is a negligible function. If we restrict* $\mathcal{A}$ *to a PPT, it is computationally binding. If the computation time is unbounded, it is statistically binding.*

2.1.4 Merkle Trees

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

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

Thanks to Merkle Trees, similarly to the TreePRG described above, it is possible to verify that some given data is correct in an easy way. 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, by revealing the sibling paths of the $(v_i)_{i \in I}$. This path is called the authentification path and is denoted $\mathsf{auth}((v_1 \ldots v_N), I)$.

2.1.5 Secret Sharing Schemes

The interactive proof relies on a prover simulating a MultiParty Computation (MPC) protocol, where each party has a share of the witness $\boldsymbol{x}$. We detail here the formalism from [FR22]. The sharing of a secret s among N parties is denoted $([\![s]\!]_1, \ldots, [\![s]\!]_N)$ where $[\![s]\!]_i$ is the share of the ith party. Given $J \subset [1, N]$, $[\![s]\!]_J$ is the subset of shares $\{[\![s]\!]_j\}_{j \in J}$.

Definition 6 (Threshold Linear Secret Sharing). *Let $\mathbb{F}$ a finite field. Let $1 < t \leq N$. A (t, N)-threshold linear secret sharing scheme (TLSSS) is a scheme to share a secret, $s \in \mathbb{F}$, in a sharing $[\![s]\!] = ([\![s]\!]_1 \ldots [\![s]\!]_N) \in \mathbb{F}^N$, where only t coordinates of $[\![s]\!]$ need to be known in order to reconstruct the secret s, while the knowledge of $t - 1$ coordinates of $[\![s]\!]$ reveals no information.*

We can write this the following way:

$$\begin{cases} \mathsf{Share} : \mathbb{F} \times R \to \mathbb{F}^N \\ \mathsf{Reconstruct}_J : \mathbb{F}^t \to \mathbb{F} \end{cases} \tag{2}$$

R corresponds to the randomness space used to build the shares. J is a subset of $\{1 \ldots N\}$, with $|J| = t$. The two algorithms of a TLSSS must satisfy some properties:

– **Correctness.** *For every $s \in \mathbb{F}, r \in R, J \subset [1, N]$ such that $|J| = t$ and for $[\![s]\!] \leftarrow \mathsf{Share}(s; r)$, we have:*

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

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

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

are perfectly indistinguishable.

– **Linearity.** *For every $v_0, v_1 \in \mathbb{F}^t, \alpha \in \mathbb{F}, J \subset [1, N]$ with $|J| = t$,*

$$\mathsf{Reconstruct}_J(\alpha \cdot v_0 + v_1) = \alpha \cdot \mathsf{Reconstruct}_J(v_0) + \mathsf{Reconstruct}(v_1).$$

We recall below two among the most used secret sharing schemes.

Definition 7 (Additive Secret Sharing). *Let $\mathbb{F}$ a field and $s \in \mathbb{F}$ a secret. An additive secret sharing with N parties is a (N, N)-threshold sharing scheme such that:*

– *$[\![s]\!]_i = r_i$ for $i \in [1, N - 1]$, where $r_i \xleftarrow{\$} \mathbb{F}$;*

– *$[\![s]\!]_N = s - \sum_{i=1}^{N-1} [\![s]\!]_i$.*

The $\mathsf{Reconstruct}_{[1,N]}$ algorithm takes as inputs all the shares, and outputs the sum of all the shares.

Definition 8 (Shamir's Secret Sharing). *Let $\mathbb{F}$ a field and $s \in \mathbb{F}$ a secret. A Shamir's secret sharing is the following $(\ell + 1, N)$-threshold sharing scheme:*

– *Sample $(r_1, \ldots, r_\ell) \xleftarrow{\$} \mathbb{F}^\ell$;*

– *Compute $P(X) = s + \sum_{i=1}^{\ell} r_i X^i$;*

– *Compute $[\![s]\!]_i = P(e_i)$ where the $(e_i)_{i \in \{1, \ldots, N\}}$ are distinct and non-zero public values.*

For J a subset of $[1, N]$ with $|J| = \ell + 1$, the $\mathsf{Reconstruct}_J$ algorithm corresponds to the interpolation of the polynomial P, when taking in inputs $[\![s]\!]_i$ for $i \in J$, and outputs the constant term, s.

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

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

Proof. See [FR22].

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

2.2 Proof of Knowledge and Digital Signature Schemes

2.2.1 Zero-Knowledge Proof of Knowledge

We define here the concept of proof of knowledge. Let $R \subset \{0, 1\}^* \times \{0, 1\}^*$ an NP-relation. $(x, \omega) \in R$ is a statement-witness pair where x is the statement and ω an associated witness. The set of valid witnesses for a statement x is $R(x) = \{\omega : (x, \omega) \in R\}$. A prover $\mathcal{P}$ wants to use a proof of knowledge to convince a verifier $\mathcal{V}$ that he knows a witness ω for a statement x.

Definition 9 (Proof of knowledge). *A proof of knowledge for a relation R with soundness ϵ is a two-party protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with a public statement x, where $\mathcal{P}$ want to convince $\mathcal{V}$ that he knows ω such that $(x, \omega) \in R$. We denote $\{\mathcal{P}(x, \omega), \mathcal{V}(x)\}$ the transcript between $\mathcal{P}$ and $\mathcal{V}$. A proof of knowledge has the following properties:*

- **Perfect completeness:** *If $(x, \omega) \in \mathcal{R}$, then a prover $\mathcal{P}$ who knows a witness ω for x succeeds in convincing the verifier $\mathcal{V}$ of his knowledge. This means that the prover convinces the verifier with probability 1, i.e,*
$$\mathsf{Pr}[\langle \mathcal{P}(x, \omega), \mathcal{V}(x)\rangle \to ACCEPT] = 1.$$

- **Soundness:** *If there exists a PPT prover $\tilde{\mathcal{P}}$ such that*
$$\tilde{\epsilon} = \mathsf{Pr}[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x)\rangle \to ACCEPT] > \epsilon,$$
then there exists an algorithm which, given rewindable black-box access to $\tilde{\mathcal{P}}$, outputs a witness ω' for x in time $\mathsf{poly}(\lambda, (\tilde{\epsilon} - \epsilon)^{-1})$ with probability at least $\frac{1}{2}$.

To be zero-knowledge while the prover interacts with a honest verifier (i.e., a verifier sending his messages according to the definition of the protocol), the PoK must verify the following property :

Definition 10 (Honest-Verifier Zero-Knowledge). *A PoK satisfies the Honest-Verifier Zero-Knowledge (HZVK) property if there exists a polynomial-time simulator 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 $\mathcal{V}(x)$.*

Definition 11 (Digital Signature Scheme). *A digital signature scheme DSS with security parameter λ is a triplet of polynomial time algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ such that:*

- *The key-generation algorithm* KeyGen *is a probabilistic algorithm which outputs a pair of keys* (pk, sk).

- *The signing algorithm* Sign, *eventually probabilistic, which takes as inputs a message m to sign and the secret key* sk, *and outputs a signature* σ.

- *The verification algorithm* Verif *which takes as inputs the public key* pk, *the message m and its signature* σ, *and outputs a bit b. The output 1 means that the signature is considered as valid.*

A correct signature scheme satisfies 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, if a signature is correctly generated, then it is always accepted.

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

Definition 12 (EUF-CMA). *We can define the following game* $G_{\text{EUF-CMA}}$ *where* $\mathsf{Sign}(sk, \cdot)$ *is an oracle that sign messages:*

$$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$$
$$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(pk)$$
$$(m, \sigma) \leftarrow \mathcal{A}$$

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

2.2.2 Fiat-Shamir Transformation

The Fiat-Shamir (FS) transformation is a generic process allowing to convert an interactive identification scheme into a signature. The main adaptation lies in the removal of the interactions in the protocol: one needs to pull the challenges in a deterministic 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 τ the number of repetitions. One will see in section 6.1 that there is an effective attack if τ is too small.

Let us describe the FS tranformation on a 5-round zero-knowledge proof of knowledge. Concretely, 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 H_1 is an hash function, $h_0^{(e)}$ is the first step commitment of the eth execution of the protocol, and salt a random value in $\{0, 1\}^{2\lambda}$. The prover obtains the first challenge (one challenge per execution $e \in [1, \tau]$) 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 H_2 and the other information computed during the step 3. The prover obtains the second challenge (one challenge per execution $e \in [1, \tau]$) from h_2 by using a XOF as well.

The signature σ therefore consists of sending:

- the salt salt;

- h_1 as commitment of the initial values;

- h_2 as hash of all responses of the first challenge;

- each response rsp of the second challenge.

The signature is:

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

2.2.3 Useful Lemmas

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

- $\mathsf{Pr}[B] \geq \alpha$,

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

- $\mathsf{Pr}[B|A] \geq \frac{\alpha}{\epsilon}$.

Proof. An interested reader can refer to [PS00].

2.2.4 q-Polynomials and Rank Metric definition

One of the main tools we are going to use are q-polynomials, as they allow us to characterize linear subspaces.

Definition 13 (q-polynomial). *A q-polynomial of q-degree r is a polynomial in $\mathbb{F}_{q^m}[X]$ of the form:*

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

Proposition 2. *Let P a q-polynomial, $\alpha, \beta \in \mathbb{F}_q$, $x, y \in \mathbb{F}_{q^m}$. We then have:*

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

Proof. This comes directly from the fact that the Frobenius endomorphism: $x \mapsto x^q$ is linear over $\mathbb{F}_q$

We can see q-polynomials as $\mathbb{F}_q$-linear applications from $\mathbb{F}_q^m$ to $\mathbb{F}_{q^m}$. It is then possible to define a linear subspace of $\mathbb{F}_{q^m}$ from a q-polynomial.

Proposition 3. *The set of roots of a non-zero q-polynomial of q-degree r forms 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, giving the upper bound on the subspace 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 every element in E is a root of P. P is called the annihilator polynomial of E.*

Let us define the usual notions of the rank metric:

Definition 14. *Let $\boldsymbol{E} \in \mathbb{F}_q^{m \times n} = \left(e_{i,j} \right)$ with $e_{i,j} \in \mathbb{F}_q$ for $(i,j) \in [1,m] \times [1,n]$, and let $\mathcal{B} = (b_1 \dots b_m)$ an $\mathbb{F}_q$-basis of $\mathbb{F}_{q^m}$. It is then possible to associate each column of $\boldsymbol{E}$ to an element of $\mathbb{F}_{q^m}$ using*

$$e_j = \sum_{i=1}^{m} b_i e_{i,j}$$

for each $j \in [1,n]$. By defining $\boldsymbol{e} = (e_1 \dots e_n)$, we can say that $\boldsymbol{e}$ is the vector associated to the matrix $\boldsymbol{E}$.

The rank weight *is defined as* $\mathrm{W}_R(\boldsymbol{e}) = \mathsf{Rank}(\boldsymbol{E})$. *The* distance *between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is then $d(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{W}_R(\boldsymbol{x} - \boldsymbol{y})$. The* support *of $\boldsymbol{e} = (e_1 \dots e_n)$ is the linear subspace of $\mathbb{F}_{q^m}$ generated by its coordinates:* $\mathsf{Supp}(\boldsymbol{e}) = \langle e_1 \dots e_n \rangle$.

Remark 1. The choice of the basis $\mathcal{B}$ does not change anything to the weight of $\boldsymbol{e}$ or the rank of $\boldsymbol{E}$. The rank of $\boldsymbol{E}$ is obviously equal to the dimension of $\mathsf{Supp}(\boldsymbol{e})$. Moreover, in the above definition, we are working on the columns of $\boldsymbol{E}$. We stress that it is possible to work with the rows instead, as it may be more efficient, depending on the values of m and n. We will also abuse notations, and sometimes note $\mathsf{Supp}(\boldsymbol{e})$ as $\mathsf{Supp}(\boldsymbol{E})$.

The number of supports possible of dimension r when working in $\mathbb{F}_{q^m}$ is

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

2.3 MPC-in-the-Head and Proof of Knowledge

The following explanation of the MPCitH paradigm comes from [Fen22, Section 2.1].

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

- the secret witness w is encoded as a sharing $[\![w]\!]$ and each $\mathcal{P}_i$ takes a share $[\![w]\!]_i$ as input;

- the function f outputs ACCEPT or REJECT;

- the views of t parties leak no information about the secret w, where $t + 1$ is the threshold of the secret sharing.

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

– she builds a random sharing $[\![w]\!]$ of w;

– she simulates locally ("in her head") all the parties of the MPC protocol;

– she sends commitments to each party's view, i.e, party's input share, secret random tape and sent and received messages, to the verifier;

– she sends the output shares $[\![f(w)]\!]$ of the parties, which should correspond to ACCEPT.

Then the verifier randomly chooses t parties and asks the prover to reveal their views. After receiving them, the verifier checks that they are consistent with an honest execution of the MPC protocol and with the commitments. Since only t parties are opened, revealed views leak no information about the secret w, while the random choice of the opened parties makes the cheating probability upper bounded by $(N - t)/N$, thus ensuring the soundness of the zero-knowledge proof.[6]

In our case, the parties take as input a linear sharing $[\![w]\!]$ of the secret w (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 ϵ from a randomness oracle $\mathcal{O}_R$. When calling this oracle, all the parties get the same random value ϵ.

Receiving hint: the parties can receive a sharing $[\![\beta]\!]$ (one share per party) from a hint oracle $\mathcal{O}_H$. The hint β 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 φ is an $\mathbb{F}$-linear function, then broadcast all the shares $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ to publicly reconstruct $\alpha := \varphi(v)$. The function φ can depend on the previous random values $\{\epsilon^i\}_i$ from $\mathcal{O}_R$ and on the previous broadcasted values. One should note that in the case of additive sharing, only one party needs to compute the addition by a constant.

We restrain here to threshold linear sharings, i.e, additive secret sharing and low-threshold linear secret sharing, which is the framework in which [FR22] and [Fen22] are set.

In [AMGH+22], another way to verify the parties' computation (in the case of additive sharing) has been introduced which is more efficient. This is the hypercube technique, which we will detail here.

The idea of the hypercube construction is the following:

– Generate $N = 2^D$ parties with an additive sharing. Those parties can be indexed either by an integer in $[1, N]$ or by a vector in $[1, 2]^D$. For example, the i-th leaf can be written as the leaf i, or as the leaf $(i_1, \ldots, i_D)$;

– For each dimension of the hypercube, compute the main shares by summing up the 2^{D-1} shares of a slice of the hypercube. We have 2 main parties per dimension, i.e, $2 \cdot D$ main parties in total. We can also establish a mapping between the main parties and the leaves of the hypercube, by writing a main party $p = (k, j) \in [1, D] \times [1, 2]$. Concretely, for a witness w, the share of the main party $p = (k, j)$ can be built (and written) $[\![w]\!]_{(k,j)} = \sum_{i:i_k=j}[\![w]\!]_i$. These main shares will be written either as $[\![w]\!]_{(k,j)}$ or as $[\![w]\!]_{(p_1,p_2)}$, depending on the indices we use.

- Execute the MPC protocol for all the dimension, i.e, execute D times the MPC protocol, each time with the 2 main parties of the dimension.

Since we built the hypercube with an additive sharing, we know that the sum of all the leaves gives us the secret. This means that we can sum them up in every way we want (the way in which we sum them depends on the dimension). This is an improvement compared to the standard MPC in the head, as instead of simulating one protocol with 256 parties, we can simulate 8 MPC protocols, with only 2 parties each. Furthermore, since the used secret is the same for every dimension and since the broadcasted plaintext values are the same for every protocol, it is possible to:

- Execute the MPC protocol for the two parties for one dimension;

- For $D - 1$ dimensions:

  - Execute the MPC protocol for one party $\mathcal{P}_1$;

  - Subtract the share broadcasted by $\mathcal{P}_1$ to the broadcast plaintext value;

  - Set the resulting share as the broadcasted share of the second party $\mathcal{P}_2$.

Even though this doesn't reduce the size of the communication cost, this is an improvement as the computations are faster (we avoid the computation for $D - 1$ parties). Moreover, we can observe that taking a hypercube with edge size larger than 2 brings no advantage over a power of 2 (see [AMGH$^+$22]). This is why N is always a power of 2 in the case of the hypercube scheme.

2.4 The MinRank Problem

Definition 15 (MinRank). *Let $\mathbb{F}_q$ be the finite field of size q, and $m, n, k, r \in \mathbb{N}^*$. The computational MinRank Problem with parameters (q, m, n, k, r) is the following problem:*
Let $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_k, \boldsymbol{E} \in \mathbb{F}_q^{m \times n}$ and $\boldsymbol{x} \in \mathbb{F}_q^k$ be uniformly sampled such that

$$\mathrm{W}_R(\boldsymbol{E}) \leq r \quad with \quad \boldsymbol{M}_0 := \boldsymbol{E} - \sum_{i=1}^{k} x_i \boldsymbol{M}_i.$$

Given $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k$, retrieve the vector $\boldsymbol{x}$.

The MinRank problem was proven to be NP-Complete by [BFS99], and plays a central role in cryptography. It is used in the attacks on HFE, and appears in syndrome decoding in rank metric. It was also used in some signatures, such as Courtois' signature [Cou01], MR-DSS [BESV22], or MinRank in the Head [ARZV22] for instance.

2.4.1 Key Generation

In practice, to generate a MinRank instance, one has to:

- Sample $\boldsymbol{x}$ uniformly in $\mathbb{F}_q^k$;

- for all $i \in [1, k]$, sample $\boldsymbol{M}_i \in \mathbb{F}_q^{m \times n}$;

- Sample $\boldsymbol{E} \in \mathbb{F}_q^{m \times n}$ such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$;

- Define $\boldsymbol{M}_0$ as $\boldsymbol{E} - \sum_{i=1}^{k} \boldsymbol{M}_i x_i$.

From this, we can quickly explicit a KeyGen algorithm:

- $\boldsymbol{x} \xleftarrow{\$} \mathbb{F}_q^k$
- For $i \in [1, k]$, $\boldsymbol{M}_i \xleftarrow{\$} \mathbb{F}_q^{m \times n}$
- $\boldsymbol{E} \xleftarrow{\$} \mathbb{F}_q^{m \times n}$ such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$
- $\boldsymbol{M}_0 = \boldsymbol{E} - \sum_{i=1}^{k} \boldsymbol{M}_i x_i$
- Set $\mathsf{pk} = (\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k)$ and $\mathsf{sk} = \boldsymbol{x}$
- Return $(\mathsf{pk}, \mathsf{sk})$.

Fig. 2: Key generation – Simple version

The size of the public key will be $m \cdot n \cdot \log_2 q + \lambda$ bits ($\boldsymbol{M}_0$ and the seed used for the random matrices), and the size of the secret key will be λ bits (if we sample $\boldsymbol{x}$ from a seed).

However, it is possible to optimize this key generation. First, we need to define the systematic form of a MinRank problem. Let $\boldsymbol{L}_1$ be the $(k+1) \times mn$ matrix, composed of $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k$ written in lines, following the row order. Then, if the first k columns and k rows of $\boldsymbol{L}_1$ form a full rank matrix, we can obtain a matrix $\boldsymbol{L} = \begin{bmatrix} \boldsymbol{I}_k & \boldsymbol{L}_1' \\ 0 \ldots 0 & \end{bmatrix}$ where $\boldsymbol{I}_k$ is the $k \times k$ identity matrix, using row operations. Row operations on the matrix $\boldsymbol{L}_1$ correspond to a linear combination of $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k$, meaning they don't change the number of solutions of the instance.

For a generic MinRank instance (i.e, with high probability, as noted in [BESV22] and used in [BBB+22]), this transformation will be possible. This means we can generate the instance directly in this systematic form, without changing the difficulty of the problem. To proceed, we will use the procedure described in [BESV22]:

- Sample $\boldsymbol{L} \in \mathbb{F}_q^{k \times mn}$ uniformly such that $\boldsymbol{L} = \begin{bmatrix} \boldsymbol{I}_k & \boldsymbol{L}' \end{bmatrix}$ where $\boldsymbol{I}_k$ is the $k \times k$ identity matrix and $\boldsymbol{L}'$ is an $k \times (mn - k)$ matrix;

- Set $\boldsymbol{M}_i$ as the i-th line of $\boldsymbol{L}$, written in a matrix form (in row order, i.e, the first row of the matrix is the first n entries of the lines, the second row is the next n entries and so on);

- Sample $\boldsymbol{E} \xleftarrow{\$} \mathbb{F}_q^{m \times n}$ uniformly such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$ and $\boldsymbol{\beta} \xleftarrow{\$} \mathbb{F}_q^k$;

- Compute $\boldsymbol{F} = \boldsymbol{E} - \sum_{i=1}^{k} \beta_i \boldsymbol{M}_i$;

- Compute $\boldsymbol{M}_0 = \boldsymbol{F} - \sum_{i=1}^{k} f_i \boldsymbol{M}_i$ where $\boldsymbol{f} = (f_1 \ldots f_k)$ are the first k entries, in row order, of $\boldsymbol{F}$;

- Set $\boldsymbol{x} = \boldsymbol{\beta} + \boldsymbol{f}$.

It is clear that we have $\boldsymbol{M}_0 = \boldsymbol{E} - \sum_{i=1}^{k} x_i \boldsymbol{M}_i$ and that the first k entries of $\boldsymbol{M}_0$ are zeros. Since $\boldsymbol{L}$ comes from a random seed, our secret key $\boldsymbol{x}$ can be retrieved from the used seed, and is thus of size λ bits. For our public key, we have $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_k$ of size λ bits, plus $\boldsymbol{M}_0$, of size $(mn - k) \cdot \log_2 q$ bits.

- Sample $\boldsymbol{L} \in \mathbb{F}_q^{k \times mn}$ uniformly such that $\boldsymbol{L}$ is of the form described above
- Set $\boldsymbol{M}_i$ as the i-th line of $\boldsymbol{L}$, written in a matrix form (in row order)
- Sample $\boldsymbol{E} \xleftarrow{\$} \mathbb{F}_q^{m \times n}$ uniformly such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$, and $\boldsymbol{\beta} \xleftarrow{\$} \mathbb{F}_q^k$
- Compute $\boldsymbol{F} = \boldsymbol{E} - \sum_{i=1}^k \beta_i \boldsymbol{M}_i$
- Compute $\boldsymbol{M}_0 = \boldsymbol{F} - \sum_{i=1}^k f_i \boldsymbol{M}_i$ where $\boldsymbol{f} = (f_1 \ldots f_k)$ are the first k entries, in row order, of $\boldsymbol{F}$
- Set $\boldsymbol{x} = \boldsymbol{\beta} + \boldsymbol{f}$
- Set $\mathsf{pk} = (\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k)$ and $\mathsf{sk} = \boldsymbol{x}$
- Return $(\mathsf{pk}, \mathsf{sk})$.

Fig. 3: Key generation – Optimized version

2.5 Rank Checking MPC protocol

It is necessary to build a protocol allowing us to verify that the space generated by a list of $n \geq r$ elements is at most r. We will build here a MPCitH ZK-proof based on Feneuil's protocol using q-polynomials [Fen22], that we will remind here. We want to check that a list of elements $(e_j)_{j \in [1,n]}$, where $e_j \in \mathbb{F}_{q^m}$, leads to a $\mathbb{F}_q$-linear subspace U of dimension at most r.

Given U, let L_U be the polynomial:

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

It is possible to show that all the e_j are roots of L_U, which is a q-polynomial. It means that L_U is of the form:

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

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

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

If one or more of the e_j is not a root of L_U, the above equation is satisfied with probability at most $\frac{1}{q^{m\eta}}$. Then,

$$\begin{aligned}
\sum_{j=1}^n \gamma_j L_U(e_j) &= \sum_{j=1}^n \gamma_j \left(\sum_{i=0}^{r-1} \beta_i e_j^{q^i} + e_j^{q^r} \right) \\
&= \sum_{j=1}^n \gamma_j e_j^{q^r} + \sum_{i=0}^{r-1} \beta_i \sum_{j=1}^n \gamma_j e_j^{q^i}
\end{aligned}$$

Defining $z = -\sum_{j=1}^n \gamma_j e_j^{q^r}$ and $w_i = \sum_{j=1}^n \gamma_j e_j^{q^i}$, proving the equation is equivalent to prove that

$$z = \langle \boldsymbol{\beta}, \boldsymbol{w} \rangle.$$

The latter equation will be checked thanks to the multiplication protocol from [BN20] (adapted in the matrix setting, see [Fen22]). In the case of MinRank, we want to apply this protocol to $\boldsymbol{E}$, which we will have to compute. In what follows, we work on the columns of $\boldsymbol{E}$ and in $\mathbb{F}_{q^m}$. Depending

on the parameters, it might also be easier to work on the rows of $\boldsymbol{E}$. This gives us the following protocol from [Fen22]:

- Public data: $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k \in \mathbb{F}_q^{m \times n}$
- Each party receives $[\![\boldsymbol{x}]\!] \in \mathbb{F}_q^k$, $[\![L]\!] = \sum_{i=0}^{r-1}[\![\beta_i]\!]X^{q^i} + X^{q^r}$ where each $\beta_i \in \mathbb{F}_{q^m}$, $[\![\boldsymbol{a}]\!]$ where $\boldsymbol{a}$ is uniformly sampled from $\mathbb{F}_{q^{m\eta}}^r$ and $[\![c]\!] \in \mathbb{F}_{q^{m\eta}}$ such that $c = -\langle \boldsymbol{a}, \boldsymbol{\beta} \rangle$.

1. The parties sample $(\gamma_1, ..., \gamma_n, \epsilon) \xleftarrow{\$} \mathbb{F}_{q^{m\eta}}^{n+1}$
2. The parties compute $[\![\boldsymbol{E}]\!] = \boldsymbol{M}_0 + \sum_{i=1}^k [\![x_i]\!]\boldsymbol{M}_i$
3. The parties set $[\![e_i]\!]$ as the element in $\mathbb{F}_{q^m}$ associated to the i-th column of $[\![\boldsymbol{E}]\!]$
4. They compute $[\![z]\!] = -\sum_{j=1}^n \gamma_j [\![e_j]\!]^{q^r}$
5. For i from 0 to $r-1$:
    - They compute $[\![w_i]\!] = \sum_{j=1}^n \gamma_j [\![e_j]\!]^{q^i}$
6. The parties compute $[\![\boldsymbol{\alpha}]\!] = \epsilon \cdot [\![\boldsymbol{w}]\!] + [\![\boldsymbol{a}]\!]$ and open the shares to get $\boldsymbol{\alpha}$
7. The parties compute $[\![v]\!] = \epsilon \cdot [\![z]\!] - \langle \boldsymbol{\alpha}, [\![\boldsymbol{\beta}]\!] \rangle - [\![c]\!]$ and open the shares to get v
8. They output ACCEPT if $v = 0$, and REJECT otherwise

Fig. 4: Protocol Π^η to check that an input is solution of an instance of MinRank

One should stress that we can locally compute shares to the power q^i. Since we are in $\mathbb{F}_q$, this operation is linear, and thus this operation is basically "free" in multiparty computation.

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

Proof. The proof is the same as the one in [Fen22, Section 5.1]

Thanks to this MPC protocol, we can build a zero-knowledge proof of knowledge protocol, using either additive secret sharing (Fig.5) or threshold secret sharing (Fig.10).

3 Description of the Protocols

3.1 Description of MIRA-Additive

3.1.1 Proof of Knowledge with Additive Secret Sharing Scheme

We describe here the idea of the hypercube technique introduced by [AMGH+22].

- The prover generates $N = 2^D$ shares of the inputs of the Π^η protocol (Fig.4), using an additive secret sharing scheme;

- He then computes the shares associated to the "main parties", i.e, the parties associated to each dimension, by summing up the 2^{D-1} shares that have the same index on the dimension (in the same way as explained in section 2.3);

- There are D dimensions, each of them having 2 main parties. The prover sends the commitments of all the N shares;

- He then receives the first challenge, which is some random values for the computation of the Π^η protocol;

– Then, for $D + 1$ main parties, he executes the protocol Π^η. For the $D - 1$ other parties, he will use the broadcasted values of $\boldsymbol{\alpha}$ and v in order to compute their shares of $\boldsymbol{\alpha}$ and of v. He will hash these values;

– He sends to the verifier the hash of these values (it is possible to hash them twice, as is done in the description of Fig.6, or not);

– He receives the second challenge, which is a single leaf, i^*;

– He reveals every commitment except the $i^{*\text{th}}$ one, and sends the share $[\![\boldsymbol{\alpha}]\!]_{i^*}$ broadcast by the party i^*;

– The verifier can then reconstruct the broadcast shares of the main parties, i.e the 2 shares for each dimension, and then check that everything is correct. Similarly to the prover, he can avoid the computation of the protocol for $D - 1$ parties.

One detail to notice is that in Fig.6, we use an hash function to output the broadcast shares. We stress that this hash function is not mandatory. The security of the protocol with or without it is the same. Depending on the implementation, using it can be more convenient (for example, to save memory when the used implementation of the hash functions is not incremental). In our security proofs, we will consider that such hash function is used.

Finally, if $i^* \neq N$, the prover has to send $[\![\boldsymbol{x}]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N$ in addition to the sibling path. This comes from the fact that these shares are not completely random.

This gives us the protocol provided in Fig.5. We insist on the fact that, as mentioned in section 2.3, a prover doesn't need to simulate $2D$ parties in step 10, but only $D + 1$, as the values of $\boldsymbol{\alpha}$ and v are the same, no matter the dimension. In fact, for $D - 1$ dimensions in step 10, it is possible to retrieve $[\![\boldsymbol{\alpha}]\!]_{(k,2)}$ by computing $\boldsymbol{\alpha} - [\![\boldsymbol{\alpha}]\!]_{(k,1)}$ instead of simulating the MPC protocol. We emphasize on this optimisation, as this is a crucial advantage of the hypercube structure.

We prove that our scheme satisfies the necessary security properties in the proof of theorem 1 in section 5.1.

3.1.2 MIRA-Additive

From the zero-knowledge protocol, we can simply deduce a signature scheme, using the Fiat-Shamir transform. The protocol is now non-interactive, and we deterministically sample the challenge thanks to a hash function. We must repeat the zero-knowledge protocol several times, in order to reach a certain level of security. Note that we use a value salt, as in [FJR22], in order to increase the security of the scheme as it reduces the probability to have seeds collisions. Moreover, simarly than in the proof of knowledge, the hash function H_3 is not mandatory and depends only of implementation choices. The signature protocol obtained is described in Fig.8 and Fig.9. As aforementioned in the proof of knowledge, the signature process avoid the computation of the MPC protocol for $D - 1$ parties in step 8. The verification process avoids it in step 6 as well.

We prove that the scheme is EUF-CMA secure in the proof of theorem 3 in section 5.2.

- Public data $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k \in \mathbb{F}_q^{m \times n}$

The prover wants to convince the verifier that he knows a solution $\boldsymbol{x} \in \mathbb{F}_q^k$ of the MinRank instance, i.e such as $\boldsymbol{E} = \boldsymbol{M}_0 + \sum_{i=1}^k (\boldsymbol{M}_i x_i)$ and $\mathrm{W}_R(\boldsymbol{E}) \leq r$

Step 1: Commitment

1. The prover sets $U = \langle e_1 \ldots e_n \rangle$, and computes $L(X) = \prod_{u \in U} (X - u) = X^{q^r} + \sum_{i=0}^{r-1} \beta_i \cdot X^{q^i}$ for some $\beta_i \in \mathbb{F}_{q^m}^r$

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

3. The prover expands the root seed recursively using TreePRG to obtain N leaves, from which he derives N seeds and N commitment random tapes $(\mathsf{seed}_{i'}, \rho_{i'})$

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

   - Sample $([\![\boldsymbol{x}]\!]_i, [\![\boldsymbol{\beta}]\!]_i, [\![\boldsymbol{a}]\!]_i [\![c]\!]_i) \xleftarrow{\$,\mathsf{seed}_i} PRG$ where PRG is a pseudo-random generator
   - $\mathsf{state}_i = \mathsf{seed}_i$

5. For the share N:

   - Sample $[\![\boldsymbol{a}]\!]_N \xleftarrow{\$,\mathsf{seed}_N} PRG$
   - Compute $[\![\boldsymbol{x}]\!]_N = \boldsymbol{x} - \sum_{i=1}^{N-1} [\![\boldsymbol{x}]\!]_i$, $[\![\boldsymbol{\beta}]\!]_N = \boldsymbol{\beta} - \sum_{i=1}^{N-1} [\![\boldsymbol{\beta}]\!]_i$ and $[\![c]\!]_N = -\langle \boldsymbol{a}, \boldsymbol{\beta} \rangle - \sum_{i=1}^{N-1} [\![c]\!]_i$
   - $\mathsf{state}_N = (\mathsf{seed}_N, [\![\boldsymbol{x}]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N)$

6. The prover computes the commitments: $\mathsf{cmt}_i = \mathsf{Com}(\mathsf{state}_i, \rho_i)$, for each $i \in [1, N]$.

7. The prover computes and sends $h_0 = \mathsf{H}(\mathsf{cmt}_1, \cdots, \mathsf{cmt}_N)$ to the verifier.

8. The prover computes the input shares of the main parties by summing all the associated leaves. We index each party by its coordinates on the hypercube: $i = (i_1, \ldots, i_D)$ where $i_k \in [1, 2]$. For all main party index $p \in \{(1,1), \ldots, (D, 2)\}$: $[\![\boldsymbol{x}]\!]_{(p_1, p_2)} = \sum_{i : i_{p_1} = p_2} [\![\boldsymbol{x}]\!]_i$, $[\![\boldsymbol{\beta}]\!]_{(p_1, p_2)} = \sum_{i : i_{p_1} = p_2} [\![\boldsymbol{\beta}]\!]_i$, $[\![\boldsymbol{a}]\!]_{(p_1, p_2)} = \sum_{i : i_{p_1} = p_2} [\![\boldsymbol{a}]\!]_i$ and $[\![c]\!]_{(p_1, p_2)} = \sum_{i : i_{p_1} = p_2} [\![c]\!]_i$.

Step 2: First Challenge

9. 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

10. For each dimension $k \in [1, D]$, the prover executes the algorithm in Fig. 6 on the set of the main parties of the dimension. This set is denoted P_k. He computes $\big(([\![\boldsymbol{\alpha}]\!]_{(k,i)}, [\![v]\!]_{(k,i)})_{i \in [1,2]}, H_k\big) \longleftarrow$ Algorithm Fig.6$\big(P_k, (\gamma_j)_{j \in [1,n]}, \epsilon\big)$. He only needs to execute the algorithm $D + 1$ times, as explained above.

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

Step 4: Second Challenge

12. The verifier gets a random leaf $i^* \in [1, N]$ and sends it to the prover.

Step 5: Second Response and Verification

13. The prover sends to the verifier: $\big(\mathsf{cmt}_{i^*}, [\![\boldsymbol{\alpha}]\!]_{i^*}, (\mathsf{state}_j, \rho_j)_{j \neq i^*}\big)$, where $(\mathsf{state}_j, \rho_j)_{j \neq i^*}$ corresponds to the sibling path to this commitment. If $i^* \neq N$, he also has to send $[\![\boldsymbol{x}]\!]_N$, $[\![\boldsymbol{\beta}]\!]_N$ and $[\![c]\!]_N$.

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

15. For all dimension $k \in [1, D]$, the verifier runs the algorithm in Fig. 7 to get $([\![\boldsymbol{\alpha}]\!]_{(k,i)}, [\![v]\!]_{(k,i)})_{i \in [1,2]}$ and H_k. Then he checks that

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

Fig. 5: MinRank Proof of Knowledge with additive-sharing MPCitH

Inputs: A set of main shares for the dimension k: $\left((\llbracket \boldsymbol{x} \rrbracket_{(k,i)}, \llbracket \boldsymbol{\beta} \rrbracket_{(k,i)}, \llbracket \boldsymbol{a} \rrbracket_{(k,i)}, \llbracket c \rrbracket_{(k,i)})\right)_{i \in [1,2]}$ and a protocol challenge $\left((\gamma_j)_{j \in [1,n]}, \epsilon\right)$

Outputs: A set of main shares $\left(\boldsymbol{\alpha}_{(k,i)}, \llbracket v \rrbracket_{(k,i)}\right)_{i \in [1,2]}$ and a commitment H_k of the execution

For each main party $i \in \{1,2\}$:

- Compute $\llbracket \boldsymbol{E} \rrbracket_{(k,i)} = \boldsymbol{M}_0 + \sum_{j=1}^{k} \llbracket x_j \rrbracket_{(k,i)} \boldsymbol{M}_j$
- Set $\llbracket e_j \rrbracket_{(k,i)}$ as the element in $\mathbb{F}_{q^m}$ associated to the j-th column of $\llbracket \boldsymbol{E} \rrbracket_{(k,i)}$
- Compute $\llbracket z \rrbracket_{(k,i)} = -\sum_{j=1}^{n} \gamma_j \cdot \llbracket e_j \rrbracket_{(k,i)}^{q^r}$

For l from 0 to $r-1$:

  $\diamond$ Compute $\llbracket w_l \rrbracket_{(k,i)} = \sum_{j=1}^{n} \gamma_j \cdot \llbracket e_j \rrbracket_{(k,i)}^{q^l}$

- Compute $\llbracket \boldsymbol{\alpha} \rrbracket_{(k,i)} = \epsilon \cdot \llbracket \boldsymbol{w} \rrbracket_{(k,i)} + \llbracket \boldsymbol{a} \rrbracket_{(k,i)}$ and reveal $\boldsymbol{\alpha}$
- Compute $\llbracket v \rrbracket_{(k,i)} = \epsilon \cdot \llbracket z \rrbracket_{(k,i)} - \langle \boldsymbol{\alpha}, \llbracket \boldsymbol{\beta} \rrbracket_{(k,i)} \rangle - \llbracket c \rrbracket_{(k,i)}$

Compute $H_k = \mathsf{H}\left((\llbracket \boldsymbol{\alpha} \rrbracket_{(k,i)}, \llbracket v \rrbracket_{(k,i)})_{i \in [1,2]}\right)$

Fig. 6: Execution of the MPC protocol Π^η on a set of main shares

---

Inputs: A leaf i^* that one does not reveal. $\tilde{i}$ is the index depending on the leaf i^*. The share $\llbracket \boldsymbol{\alpha} \rrbracket_{i^*}$, and all the main parties shares $\left((\llbracket \boldsymbol{x} \rrbracket_{(k,i)}, \llbracket \boldsymbol{\beta} \rrbracket_{(k,i)}, \llbracket \boldsymbol{a} \rrbracket_{(k,i)}, \llbracket c \rrbracket_{(k,i)})\right)_{i \in [1,2]}$. For $i = \tilde{i}$, the main shares correspond to the usual main party shares, except the share i^* is missing.

Outputs: $(\llbracket \boldsymbol{\alpha} \rrbracket_{(k,i)}, \llbracket v \rrbracket_{(k,i)})_{i \in [1,2]}$ and a commitment H_k of the execution

For $i \in [1,2]$:

  If $i = \tilde{i}$:

  Set $\llbracket v \rrbracket_{i^*}$ such that $v = 0$ and compute $\llbracket v \rrbracket_{(k,\tilde{i})}$

  Compute $\llbracket \boldsymbol{\alpha} \rrbracket_{(k,\tilde{i})} = \sum_{j : j_k = \tilde{i}, j \neq i^*} \llbracket \boldsymbol{\alpha} \rrbracket_j + \llbracket \boldsymbol{\alpha} \rrbracket_{i^*}$ where $\sum_{j : j_k = \tilde{i}, j \neq i^*} \llbracket \boldsymbol{\alpha} \rrbracket_j$ is obtained by applying Π^η to the partially-aggregated input shares.

  Else:

  Do the same computations as in Π^η to obtain the correct shares $\llbracket \boldsymbol{\alpha} \rrbracket_{(k,i)}$ and $\llbracket v \rrbracket_{(k,i)}$

Compute $H_k = \mathsf{H}\left((\llbracket \boldsymbol{\alpha} \rrbracket_{(k,i)}, \llbracket v \rrbracket_{(k,i)})_{i \in [1,2]}\right)$

Fig. 7: Check the executions of the MPC protocol Π^η on the main parties

Inputs

- Secret key $\mathsf{sk} = (\boldsymbol{x}) \in \mathbb{F}_q^k$, $(\boldsymbol{E}) \in \mathbb{F}_q^{m \times n}$ such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$
- Public data $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k \in \mathbb{F}_q^{m \times n}$ with $\boldsymbol{E} = \boldsymbol{M}_0 + \sum_{i=1}^{k} \boldsymbol{M}_i x_i$
- Message $m \in \{0,1\}^*$

Step 1: Commitment

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

2. Set $U = \langle e_1, \ldots, e_n \rangle$ and compute $\boldsymbol{\beta} = (\beta_k)_{k \in [0, r-1]} \in \mathbb{F}_{q^m}^r$ the coefficients of the annihilator q-polynomial $L(X)$ associated to $\boldsymbol{E}$ such that $L(X) = \prod_{u \in U}(X - u) = X^{q^r} + \sum_{k=0}^{r-1} \beta_k \cdot X^{q^k}$ and $\forall j \in [1,n], L(e_j) = 0$ where each e_j is the element in $\mathbb{F}_{q^m}$ associated to the j-th column of $\boldsymbol{E}$

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

   - Sample a root seed: $\mathsf{seed}^{(e)} \leftarrow \{0,1\}^\lambda$

   - Expand root seed recursively with TreePRG to obtain N seeds and randomness $(\mathsf{seed}_{i'}^{(e)}, \rho_{i'})$

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

    If $i \neq N$:

     $\diamond$ Sample $([\![\boldsymbol{x}]\!]_i^{(e)}, [\![\boldsymbol{\beta}]\!]_i^{(e)}, [\![\boldsymbol{a}]\!]_i^{(e)} [\![c]\!]_i^{(e)}) \xleftarrow{\$, \mathsf{seed}_i^{(e)}} PRG$ where PRG is a pseudo-random generator

     $\diamond$ $\mathsf{state}_i^{(e)} = \mathsf{seed}_i^{(e)}$

    Else:

     $\diamond$ Sample $[\![\boldsymbol{a}]\!]_N^{(e)} \xleftarrow{\$, \mathsf{seed}_N^{(e)}} PRG$

     $\diamond$ Compute $[\![\boldsymbol{x}]\!]_N^{(e)} = \boldsymbol{x} - \sum_{i=1}^{N-1}[\![\boldsymbol{x}]\!]_i^{(e)}$, $[\![\boldsymbol{\beta}]\!]_N^{(e)} = \boldsymbol{\beta} - \sum_{i=1}^{N-1}[\![\boldsymbol{\beta}]\!]_i^{(e)}$ and $[\![c]\!]_N^{(e)} = -\langle \boldsymbol{a}, \boldsymbol{\beta} \rangle - \sum_{i=1}^{N-1}[\![c]\!]_i^{(e)}$

     $\diamond$ $\mathsf{state}_N^{(e)} = (\mathsf{seed}_N^{(e)}, [\![\boldsymbol{x}]\!]_N^{(e)}, [\![\boldsymbol{\beta}]\!]_N^{(e)}, [\![c]\!]_N^{(e)})$

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

   4. For all main party index $p = (p_1, p_2) \in \{(1,1), \ldots (D,2)\}$: $[\![\boldsymbol{x}]\!]_{(p_1,p_2)} = \sum_{i:i_{p_1}=p_2}[\![\boldsymbol{x}]\!]_i$, $[\![\boldsymbol{\beta}]\!]_{(p_1,p_2)} = \sum_{i:i_{p_1}=p_2}[\![\boldsymbol{\beta}]\!]_i$, $[\![\boldsymbol{a}]\!]_{(p_1,p_2)} = \sum_{i:i_{p_1}=p_2}[\![\boldsymbol{a}]\!]_i$ and $[\![c]\!]_{(p_1,p_2)} = \sum_{i:i_{p_1}=p_2}[\![c]\!]_i$.

   5. Compute and commit $h_0^{(e)} = \mathsf{H}_1(\mathsf{salt}, e, \mathsf{cmt}_1^{(e)}, \ldots, \mathsf{cmt}_N^{(e)})$

   6. Compute $h_1 = \mathsf{H}_2\left(\mathsf{salt}, m, h_0^{(1)}, \ldots, h_0^{(\tau)}\right)$

Step 2: First Challenge

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

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.6 to obtain $\left(([\![\boldsymbol{\alpha}^{(e)}]\!]_{(k,i)})_{i \in [1,2]}, ([\![v^{(e)}]\!]_{(k,i)})_{i \in [1,2]}\right)$ and $H_k^{(e)}$, with the hash function H_3

9. Compute $h_2 = \mathsf{H}_4(m, \mathsf{pk}, \mathsf{salt}, h_1, ([\![H_1^{(e)}]\!] \ldots [\![H_D^{(e)}]\!])_{e \in [1,\tau]})$

Step 4: Second Challenge

10. Sample $i^{*(e)} \xleftarrow{\$} PRG(h_2)$ where $i^{*(e)} \in [1, N]^\tau$

Step 5: Second Response

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

   - Compute $\mathsf{rsp}^{(e)} = \left((\mathsf{state}_j)_{j \neq i^*}^{(e)}, \mathsf{cmt}_{i^{*(e)}}, [\![\boldsymbol{\alpha}^{(e)}]\!]_{i^{*(e)}}\right)$ where $(\mathsf{state}_j)_{j \neq i^*}^{(e)}$ corresponds to the sibling path. If $i^* \neq N$, he also has to send $[\![\boldsymbol{x}]\!]_N, [\![\boldsymbol{\beta}]\!]_N$ and $[\![c]\!]_N$.

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

Fig. 8: MIRA Signature Scheme based on additive sharing MPCitH - Signature Algorithm

Inputs

- Public data $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k \in \mathbb{F}_q^{m \times n}$ such that there is $\boldsymbol{x} \in \mathbb{F}_q^k$ such that $\boldsymbol{E} = \boldsymbol{M}_0 + \sum_{i=1}^{k}(\boldsymbol{M}_i x_i)$ and $\mathrm{W}_R(\boldsymbol{E}) \leq r$
- Message $m \in \{0,1\}^*$
- Signature $\sigma = (\mathsf{salt}, h_1, h_2, (\mathsf{r\bar{s}p}^{(e)})_{e \in [1,\tau]})$

Step 1: Parse signature

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

2. Sample $i^{*(e)} \xleftarrow{\$} PRG(h_2)$ where $i^{*(e)} \in [1, N]$

3. Parse $\left((\mathsf{state}_j)_{j \neq i^*}^{(e)}, \mathsf{cmt}_{i^{*(e)}}, [\![\boldsymbol{\alpha}^{(e)}]\!]_{i^{*(e)}}\right) \leftarrow \mathsf{rsp}^{(e)}$

Step 2: Recompute h_1

4. **For** each iteration $e \in [1, \tau]$:
   - **For** each $i \neq i^{*(e)}$:
     - $\diamond$ $\mathsf{cmt}_i^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{(e)})$
   - Compute $h_0^{(e)} = \mathsf{H}_1\left(\mathsf{salt}, e, \mathsf{cmt}_1^{(e)}, ..., \mathsf{cmt}_N^{(e)}\right)$

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

Step 3: Recompute h_2

6. **For** each iteration $e \in [1, \tau]$:
   - Simulate MPC protocol Π^η on main parties.
   - **For** each dimension $k \in [1, D]$:
     - $\diamond$ Run the algorithm in Fig. 7 to get $H_k^{\overline{(e)}}$, in which $[\![\boldsymbol{\alpha}^{(e)}]\!]_{i^{*(e)}}$ is used to compute the share of $\boldsymbol{\alpha}$ of the main party relying on i^*.

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

Step 4: Verify signature

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

Fig. 9: MIRA Signature Scheme based on additive MPCitH - Verification Algorithm

3.2 Description of MIRA-Threshold

3.2.1 Proof of Knowledge with Threshold Secret Sharing

Once we have this zero-knowledge protocol (and signature scheme) with additive sharing, one may wonder if it is possible to use other secret sharing schemes to build MPCitH-based signature schemes. The answer is positive and given in [FR22]: we can use any threshold linear secret sharing scheme. (Ligero [AHIV17] already considered Shamir's secret sharing but this scheme is only interesting for "medium-size circuits", which is not the case of circuits involved to build signature schemes.) When dealing with threshold secret sharing, we will consider (except when told otherwise) that we are referring to Shamir's secret sharing scheme. As reminded earlier (see Definition 8), in order to build a sharing of a secret in this setting, one has to build the random polynomial with the secret as the constant term, and then evaluate this polynomial in the evaluation points. Since most of our secrets in our case are vectors, this has to be done for every coordinates, and so the notation can be pretty heavy. For the sake of simplicity, this process is not made explicit in the description of our protocols.

The protocol with threshold secret sharing works in the following way:

- Using a $(\ell+1, N)$-threshold secret sharing scheme, the prover computes the shares of the inputs of the protocol Π^η, computes the commitments, and then computes the Merkle tree root of the commitments;

- He then receives the first challenge, i.e, the randomness for the protocol Π^η;

- He then chooses a set S of $\ell + 1$ parties, for which he executes the protocol Π^η, and computes the hash value of the executions;

- He then receives the second challenge. This challenge is a set I of ℓ parties (instead of just 1 as in the additive protocol). For these ℓ parties, he sends the authentication path of the commitments, and the elements to rebuild their shares. He then chooses one party, i^*, in $S \setminus I$, and reveal the value $[\![\boldsymbol{\alpha}]\!]_{i^*}$ computed for this party;

- Thanks to this value, the verifier can recover everything he needs, and can then authenticate, or not, the prover.

Note that the knowledge of ℓ shares provides no information on the secret, since we use a $(\ell+1, N)$-threshold secret sharing. Moreover, this method permits to reduce the number of operations, for the cost a larger signature size (we refer to [FR22] for more details).

However, one limitation arises: when using Shamir's secret sharings (or any low-threshold LSSS), we need to have $N \leq q$ to share values of $\mathbb{F}_q$. Finally, this gives us the proof of knowledge of Fig.10.

Let us look at the soundness of the protocol. If the probability of a false positive of the MPC protocol was equal to 0 (i.e, the probability of a false positive of Π^η was 0), then a malicious prover (who doesn't know a solution of the MinRank instance) would need to cheat for exactly $N - \ell$ parties. Indeed, if he cheated for less parties, then at least $\ell+1$ shares would be consistent. Since we suppose the probability of false-positive is 0, this means he would know a good witness. If he cheated for more than $N - \ell$ parties, the verifier would always ask for the opening of a corrupted party, hence he would always discover the cheat. This means that the malicious prover needs to cheat on exactly $N - \ell$ parties, and the only way that the verifier is convinced is when the verifier asks the opening of the exact set of the ℓ honest parties. This means that the probability that the verifier does not

- Public data $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k \in \mathbb{F}_q^{m \times n}$

The prover wants to convince the verifier that he knows a solution $\boldsymbol{x} \in \mathbb{F}_q^k$ of the MinRank instance, i.e such as $\boldsymbol{E} = \boldsymbol{M}_0 + \sum_{i=1}^k \boldsymbol{M}_i x_i$ and $\mathrm{W}_R(\boldsymbol{E}) \leq r$

Step 1: Commitment

1. The prover builds a set of shares into a $(\ell + 1, N)$−threshold secret sharing of $\boldsymbol{x}$. The prover also builds shares of the annihilator polynomial $L(X) = X^{q^r} + \sum_{i=0}^{r-1} \beta_i \cdot X^{q^i}$ by setting $U = \langle e_1 \ldots e_n \rangle$, where $\boldsymbol{e} = (e_1, \ldots, e_n)$ is the vector associated to $\boldsymbol{E} = \boldsymbol{M}_0 + \sum_{i=1}^k \boldsymbol{M}_i x_i$, and computing $L(X) = \prod_{u \in U}(X - u) = X^{q^r} + \sum_{i=0}^{r-1} \beta_i (X^{q^i})$. He also builds 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.

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

   - $\mathsf{state}_i = (\llbracket \boldsymbol{x} \rrbracket_i, \llbracket \boldsymbol{\beta} \rrbracket_i, \llbracket \boldsymbol{a} \rrbracket_i \llbracket c \rrbracket_i)$

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

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

Step 2: First Challenge

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

Step 3: First Response

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

7. The prover executes the algorithm described in Fig.11 for the shares $i \in S$, and obtains $(\llbracket \boldsymbol{\alpha} \rrbracket_i, \llbracket v \rrbracket_i)_{i \in S}$.

8. The prover computes the value $h_1 = \mathsf{H}\left((\llbracket \boldsymbol{\alpha} \rrbracket_i, \llbracket v \rrbracket_i)_{i \in S}\right)$

Step 4: Second Challenge

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

Step 5: Second Response

10. 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 $\tilde{h}_0$.

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

12. The verifier computes $\tilde{h}_1$ by choosing $\llbracket v \rrbracket_{i^*}$ such that $v = 0$.

13. The verifier outputs ACCEPT if $(\tilde{h}_0, \tilde{h}_1) = (h_0, h_1)$, REJECT otherwise.
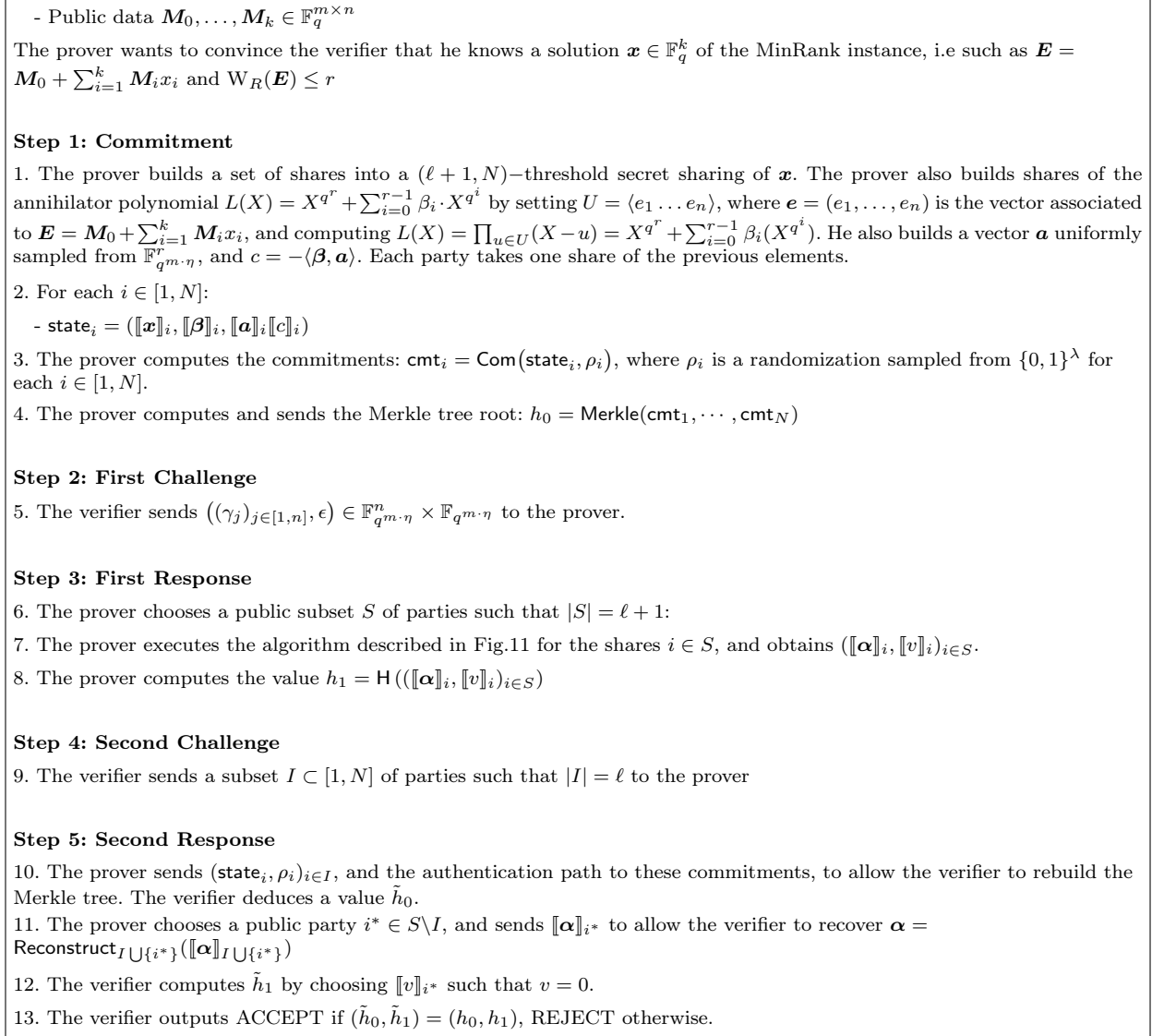
Fig. 10: MinRank Proof of Knowledge based on Threshold MPCitH

detect the cheat is $\frac{1}{\binom{N}{\ell}}$.

However, p is not 0 in our case. We can make the same reasoning, and furthermore, consider this false positive rate. We would expect to get a probability for a malicious prover to convince a verifier of $\frac{1}{\binom{N}{\ell}} + (1 - \frac{1}{\binom{N}{\ell}}) \cdot p$ (We remind the reader that $p = \frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}$).

Unfortunately, it is just a lower bound of the soundness of the protocol. As remarked in [FR22], there is a more effective way for a malicious prover to cheat at the protocol, as we will see in the proof of the theorem 2 in section 5.1.

3.2.2 MIRA-Threshold

Fig. 11: Execution of the MPC protocol Π^η on a set of shares

From the zero-knowledge protocol, we can deduce a signature scheme thanks to the Fiat-Shamir transformation. The resulting scheme is non-interactive and we deterministically sample the challenge using a hash function. We must repeat the zero-knowledge protocol several times (τ times), in order to achieve a certain level of security. The obtained signature scheme is described in Fig.12 and Fig.13.

We prove that the scheme is EUF-CMA secure in the proof of theorem 4 in 5.2.

Inputs

- Public data $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k \in \mathbb{F}_q^{m \times n}$

- Secret key $\mathsf{sk} = (\boldsymbol{x}) \in \mathbb{F}_q^k$, $(\boldsymbol{E}) \in \mathbb{F}_q^{m \times n}$ such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$ and $\boldsymbol{E} = \boldsymbol{M}_0 + \sum_{i=1}^k \boldsymbol{M}_i x_i$

- Message $m \in \{0, 1\}^*$

Step 1: Commitment

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

2. Compute $\boldsymbol{\beta} = (\beta_k)_{k \in [0, r-1]} \in \mathbb{F}_{q^m}^r$ the coefficients of the annihilator q-polynomial $L(X)$ associated to $\boldsymbol{E}$ such that $L(X) = X^{q^r} + \sum_{k=0}^{r-1} \beta_k (X^{q^k})$ and $\forall j \in [1, n], L(\boldsymbol{e}_j) = 0$ where each $\boldsymbol{e_j}$ is the element in $\mathbb{F}_{q^m}$ associated to the j-th column of E

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

  - 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 $c^{(e)} = -\langle \boldsymbol{\beta}, \boldsymbol{a}^{(e)} \rangle$

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

    $\diamond$ Compute the $(\ell + 1, N)$-threshold sharings $[\![\boldsymbol{x}^{(e)}]\!], [\![\boldsymbol{\beta}^{(e)}]\!], [\![\boldsymbol{a}^{(e)}]\!], [\![c^{(e)}]\!]$ of $\boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{a}^{(e)}$ and $c^{(e)}$

    $\diamond$ Compute $\mathsf{state}_i^{(e)} = ([\![\boldsymbol{x}^{(e)}]\!]_i, [\![\boldsymbol{\beta}^{(e)}]\!]_i, [\![\boldsymbol{a}^{(e)}]\!]_i, [\![c^{(e)}]\!]_i)$ and $\mathsf{cmt}_i^{(e)} = \mathsf{H}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{(e)})$

  - 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. Sample $\big((\gamma_j^{(e)})_{j \in [1, n]}, \epsilon^{(e)}\big)_{e \in [1, \tau]} \xleftarrow{\$} PRG(h_1)$ where $\big((\gamma_j^{(e)})_{j \in [1, n]}, \epsilon^{(e)}\big)_{e \in [1, \tau]} \in (\mathbb{F}_{q^{m \cdot \eta}}^n \times \mathbb{F}_{q^{m \cdot \eta}})^\tau$

Step 3: First Response

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

  - For each party $i \in S$ with S a public subset of parties such that $|S| = \ell + 1$:

    $\diamond$ Compute $[\![\boldsymbol{E}^{(e)}]\!]_i = \boldsymbol{M_0} + \sum_{j=1}^k \boldsymbol{M_j} [\![x_j]\!]_i$ and set $[\![e_j]\!]_i$ as the element in $\mathbb{F}_{q^m}$ associated to the j-th column of $[\![\boldsymbol{E}]\!]$

    $\diamond$ Compute $[\![z^{(e)}]\!]_i = -\sum_{j=1}^n \gamma_j \cdot [\![e_j^{(e)}]\!]_i^{q^r}$ and $\forall k \in [0, r-1], [\![w_k^{(e)}]\!]_i = \sum_{j=1}^n \gamma_j \cdot [\![e_j^{(e)}]\!]_i^{q^k}$

    $\diamond$ Compute $[\![\boldsymbol{\alpha}^{(e)}]\!]_i = \epsilon^{(e)} \cdot [\![\boldsymbol{w}^{(e)}]\!]_i + [\![\boldsymbol{a}^{(e)}]\!]_i$ and $[\![v^{(e)}]\!]_i = \epsilon^{(e)} \cdot [\![z^{(e)}]\!]_i - \langle \boldsymbol{\alpha}^{(e)}, [\![\boldsymbol{\beta}^{(e)}]\!]_i \rangle - [\![c^{(e)}]\!]_i$

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

Step 5: Second Response

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

  - Choose deterministically a party $i^{*(e)} \in S \setminus I^{(e)}$ and compute $[\![\boldsymbol{\alpha}^{(e)}]\!]_{i^{*(e)}}$

  - Compute the authentication path $\mathsf{auth}^{(e)}$ associated to root $h_0^{(e)}$ and $(\mathsf{cmt}_i^{(e)})_{i \in I^{(e)}}$

  - Compute $\mathsf{rsp}^{(e)} = \big(([\![\boldsymbol{x}^{(e)}]\!]_i, [\![\boldsymbol{\beta}^{(e)}]\!]_i, [\![\boldsymbol{a}^{(e)}]\!]_i, [\![c^{(e)}]\!]_i)_{i \in I^{(e)}}, \mathsf{auth}^{(e)}, [\![\boldsymbol{\alpha}^{(e)}]\!]_{i^{*(e)}}\big)$

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

Fig. 12: MIRA Signature Scheme based on Threshold MPCitH - Signature Algorithm

Inputs

- Public data $M_0, \ldots, M_k \in \mathbb{F}_q^{m \times n}$ such that there is $x \in \mathbb{F}_q^k$ such that $E = M_0 + \sum_{i=1}^{k}(M_i x_i)$ and $\mathrm{W}_R(E) \leq r$
- 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 $\left((\gamma_j^{(e)})_{j \in [1,n]}, \epsilon^{(e)}\right)_{e \in [1,\tau]} \xleftarrow{\$} PRG(h_1)$ where $\left((\gamma_j^{(e)})_{j \in [1,n]}, \epsilon^{(e)}\right)_{e \in [1,\tau]} \in (\mathbb{F}_{q^{m \cdot \eta}}^n \times \mathbb{F}_{q^{m \cdot \eta}})^\tau$

2. Sample $(I^{(e)})_{e \in [1,\tau]} \xleftarrow{\$} PRG(h_2)$ where $(I^{(e)})_{e \in [1,\tau]} \in (\{I \subset N \mid |I| = \ell\})^\tau$

3. For each iteration $e \in [1, \tau]$:
   - Choose deterministically $i^{*(e)}$ from $S \setminus I^{(e)}$
   - Parse $\bar{\mathsf{rsp}}^{(e)} := \left(([\![\bar{x}^{(e)}]\!]_i, [\![\bar{\beta}^{(e)}]\!]_i, [\![\bar{a}^{(e)}]\!]_i, [\![\bar{c}^{(e)}]\!]_i)_{i \in \bar{I}^{(e)}}, \bar{\mathsf{auth}}^{(e)}, [\![\bar{\alpha}^{(e)}]\!]_{i^{*(e)}}\right)$

Step 1: Recompute h_1

4. For each iteration $e \in [1, \tau]$:
   - For each party $i \in I^{(e)}$:
     ◇ Compute $\bar{\mathsf{state}}_i^{(e)} = ([\![\bar{x}^{(e)}]\!]_i, [\![\bar{\beta}^{(e)}]\!]_i, [\![\bar{a}^{(e)}]\!]_i, [\![\bar{c}^{(e)}]\!]_i)$ 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 h_2

6. For each iteration $e \in [1, \tau]$:
   - For each party $i \in I^{(e)}$:
     ◇ Compute $[\![E^{(e)}]\!]_i = M_0 + \sum_{j=1}^{k} M_j [\![x_j]\!]_i$ and set $[\![e_j]\!]_i$ as the element in $\mathbb{F}_{q^m}$ associated to the j-th column of $[\![E]\!]$
     ◇ Compute $[\![z^{(e)}]\!]_i = -\sum_{j=1}^{n} \gamma_j \cdot [\![e_j^{(e)}]\!]_i^{q^r}$ and $\forall k \in [0, r-1], [\![w_k^{(e)}]\!]_i = \sum_{j=1}^{n} \gamma_j \cdot [\![e_j^{(e)}]\!]_i^{q^k}$
     ◇ Compute $[\![\alpha^{(e)}]\!]_i = \epsilon^{(e)} \cdot [\![w^{(e)}]\!]_i + [\![a^{(e)}]\!]_i$
   - Reconstruct $\bar{\alpha}^{(e)}$ and $([\![\bar{\alpha}^{(e)}]\!]_i)_{i \in S}$ from $([\![\bar{\alpha}^{(e)}]\!]_i)_{i \in I^{(e)}}$ and $[\![\bar{\alpha}^{(e)}]\!]_{i^{*(e)}}$

7. For each iteration $e \in [1, \tau]$:
   - For each party $i \in I^{(e)}$:
     ◇ Compute $[\![\bar{v}^{(e)}]\!]_i = \epsilon^{(e)} \cdot [\![\bar{z}^{(e)}]\!]_i - \langle \bar{\alpha}^{(e)}, [\![\bar{\beta}^{(e)}]\!]_i \rangle - [\![\bar{c}^{(e)}]\!]_i$
   - Reconstruct $([\![\bar{v}^{(e)}]\!]_i)_{i \in S}$ from $([\![\bar{v}^{(e)}]\!]_i)_{i \in I^{(e)}}$ and $\bar{v}^{(e)} = 0$

8. Compute $\bar{h}_2 = \mathsf{H}_2(m, \mathsf{pk}, \mathsf{salt}, \bar{h}_1, ([\![\bar{\alpha}^{(e)}]\!]_i, [\![\bar{v}^{(e)}]\!]_i)_{i \in S, e \in [1,\tau]})$

Step 5: Verify signature

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

Fig. 13: MIRA Signature Scheme based on Threshold MPCitH - Verification Algorithm

4 Parameter Sets

4.1 Parameters Choice

Our signature scheme uses the following parameters:

– the power of a prime number, $q \in \mathbb{N}$, to build $\mathbb{F}_q$;

– a positive integer, $m \in \mathbb{N}$, the number of rows of our matrices;

– a positive integer, $n \in \mathbb{N}$, the number of columns of our matrices;

– a positive integer, $k \in \mathbb{N}$, the length of the secret vector $\boldsymbol{x}$, and $k+1$ is the number of matrices in the public key;

– a positive integer, $r \in \mathbb{N}$, the rank of the matrix $\boldsymbol{E}$;

– a positive integer, $N \in \mathbb{N}$, the number of parties simulated in the MPC protocol;

– a positive integer, $\eta \in \mathbb{N}$, to build $\mathbb{F}_{q^{m \cdot n}}$;

– a positive integer, $\tau \in \mathbb{N}$, the number of rounds in the signature.

In the LSSS-based scheme, there is also a parameter $\ell \in \mathbb{N}$, which is the value of the privacy threshold of the used linear secret sharing scheme.

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

– The security of the MinRank instance, *i.e.* the complexity of the attacks on the chosen MinRank parameters;

– The security of the signature scheme, *i.e.* the cost of the best forgery attack;

– The size of the signature.

We propose the following parameters for the MinRank instance, for the security level which corresponds to the NIST Security Level 1:

– For the additive-based scheme: $(q, m, n, k, r) = (16, 16, 16, 120, 5)$;

– For the LSSS-based scheme: $(q, m, n, k, r) = (251, 12, 13, 55, 5)$.

In the first scheme, we take the value of $q = 16$, since it leads to almost the shortest signature size and has the advantage to be easily serialized (two elements of $\mathbb{F}_q$ can be stored in a byte). However, in the second scheme, we take $q = 251$ since we have the constraint $N \leq q$ with Shamir's secret sharing (since each share is associated to a distinct field element, we can not have more shares than the size of the field). Since the protocol is more efficient when N is large, we need to take a larger value of q in the threshold-based scheme compared to the additive-based scheme. The parameters (m, n, k, r) are then chosen such that the MinRank instances are secure against the existing attacks, that we present shortly in section 6.2, and such that $k + 1 = (m - r)(n - r)$, as it corresponds exactly to the Gilbert-Varshamov bound.

Finally, we need to choose τ and η such that our signature scheme resists to the forgery attack described by [KZ20]. More precisely, in the additive-based scheme, we select them such that

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

is higher than 2^λ, with $p = \frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}$. In the case of the threshold protocol, this formula becomes:

$$\text{cost}_{\text{forge}} = \min_{0 \le \tau' \le \tau} \left\{ \frac{1}{\sum_{i=\tau'}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i}} + \binom{N}{\ell}^{\tau - \tau'} \right\} \tag{4}$$

with $p = (\frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}) \cdot \binom{N}{\ell+1}$. The factor $\binom{N}{\ell+1}$ in the value of p comes from the fact that an attacker can commit an invalid sharing of the MPC inputs in the first step of the scheme (see [FR22] for details), and the addition of $\binom{N}{\ell}^{\tau - \tau'}$ comes from the fact that there are $\binom{N}{\ell}$ possible second challenges.

In any case, as long as the forgery cost is high enough, we can just take the parameters which lead to the shortest signatures. To proceed, we need to compute the theoretical size of the signature, which we will do in the following section.

4.2 Key and Signature Sizes

4.2.1 Additive Protocol Signature Size

For the additive-sharing protocol (Fig.8), we have to send τ times the following elements:

- $com_{i^*} \in \{0,1\}^{2\lambda}$;
- $(state_{i'}^{(e)})_{i' \ne i^*}$;
- $[\![\boldsymbol{\alpha}]\!] \in \mathbb{F}_{q^{m\eta}}^r$;
- $[\![\boldsymbol{x}]\!] \in \mathbb{F}_q^k$;
- $[\![\boldsymbol{\beta}]\!] \in \mathbb{F}_{q^m}^r$;
- $[\![c]\!] \in \mathbb{F}_{q^{m\eta}}$.

Note that for each $e \in [1, \tau]$, we do not send all the $N - 1$ states $(state_{i'}^{(e)})_{i' \ne i^*}$. To reveal the state of all the parties except one, the prover only need to send the sibling path of the hidden leaf party in the seed tree of TreePRG. Thus, the number of revealed seeds in $\{0,1\}^\lambda$ is equal to the depth of the tree, which is $D = \log_2 N$. We also add h_1, h_2, and $\mathsf{salt} \in \{0,1\}^{2\lambda}$, which add up to 6λ. This gives us the following signature size (in bits):

$$|\sigma| = \underbrace{6\lambda}_{\mathsf{salt},\, h1,\, h2} + \tau \cdot \left(\left(\underbrace{k}_{\boldsymbol{x}} + \underbrace{r \times m}_{\boldsymbol{\beta}} + \underbrace{(r+1) \times m \times \eta}_{\boldsymbol{\alpha} \text{ and } c} \right) \cdot \log_2 q + \underbrace{2\lambda + D\lambda}_{\text{additive MPCitH}} \right).$$

We then obtain the following sizes:

NIST Security Level	q	m	n	k	r	N	τ	η	Public Key size (Bytes)	Secret Key size (Bytes)	Signature size (Bytes)
1	16	16	16	120	5	256	18	1	84	16	5.640
3	16	19	19	168	6	256	26	1	121	24	11.779
5	16	23	22	271	6	256	34	1	150	32	20.762

4.2.2 Threshold Protocol Signature Size

For the threshold sharing protocol (Fig.12), we have to send τ times the following elements:

- ℓ times $[\![x]\!] \in \mathbb{F}_q^k$;
- ℓ times $[\![\beta]\!] \in \mathbb{F}_{q^m}^r$;
- ℓ times $[\![a]\!] \in \mathbb{F}_{q^m \cdot \eta}^r$;
- ℓ times $[\![c]\!] \in \mathbb{F}_{q^m}$;
- $[\![\alpha]\!] \in \mathbb{F}_{q^m \cdot \eta}^r$;
- the authentification path auth.

Let us remark that we cannot use a TreePRG for the threshold scheme, since the shares of a low-threshold sharing are correlated.

For the authentification path, we do not need to send all the states. Indeed, to allow the verification of ℓ states, the prover sends their sibling paths which at most contain $\ell \cdot \log_2(\frac{N}{\ell})$ labels in total (each of 2λ bits). In the sizes we will give, we are going to use the average cost (with low deviation) of sending the sibling paths. We also send h_1, h_2, and $\mathsf{salt} \in \{0,1\}^{2\lambda}$, which add up to 6λ. We find the following signature size (in bits):

$$|\sigma| \le \underbrace{6\lambda}_{\mathsf{salt}, h1, h2} + \tau \cdot \left(\left(\ell \cdot (\underbrace{k}_{x} + \underbrace{r \times m}_{\beta} + \underbrace{(r+1) \times m \times \eta}_{a \text{ and } c}) + \underbrace{r \times m \times \eta}_{\alpha} \right) \cdot \log_2(q) + \underbrace{2\lambda \cdot \ell \log_2(\frac{N}{\ell})}_{\text{Threshold MPCitH}} \right).$$

We then obtain the following sizes:

NIST Security Level	q	m	n	k	r	N	ℓ	τ	η	Public Key size (Bytes)	Secret Key size (Bytes)	Signature size (Bytes)
1	251	12	13	55	5	251	3	7	1	117	16	8.318
3	251	16	15	109	5	251	3	10	1	155	24	17.797
5	251	16	17	109	6	251	3	14	1	195	32	30.381

5 Security Analysis

5.1 Security Proofs for the Proofs of Knowledge

5.1.1 Security Proofs for the Additive Protocol

We first need to prove that the proof of knowledge is sound and zero-knowledge:

Theorem 1. *The MinRank Proof of Knowledge protocol based on additive secret sharing described in Fig.5 has the following properties:*

- **Completeness:** *A prover $\mathcal{P}$ who has the knowledge of a solution of a MinRank instance will always be accepted by the verifier.*

- **Soundness:** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that convinces the verifier to accept with probability*

$$\tilde{\epsilon} > \epsilon$$

with

$$\epsilon = \frac{1}{N} + p \cdot (1 - \frac{1}{N})$$

where ϵ is the soundness of the protocol in Fig.5, and p is the false positive rate of the MPC protocol used, i.e, $\frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}$.
Then, there is an efficient probabilistic extraction algorithm, $\mathcal{E}$ that, given a rewindable black-box access to $\tilde{\mathcal{P}}$, outputs either a solution of the MinRank instance, or a commitment collision by making a number of calls to $\tilde{\mathcal{P}}$ which is bounded by

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

- **Honest-Verifier Zero-Knowledge** *If the pseudo-random generator algorithm PRG and the commitment Com are indistinguishable from the uniform random distribution, then the algorithm 5 is Honest-Verifier Zero Knowledge.*

Proof.
Completeness:
By construction, if the prover has knowledge of a solution of the MinRank instance, he will always be able to execute the protocol correctly, i.e, he will always obtain $\boldsymbol{\alpha}$ such that $v = 0$ when executing the MPC protocol Π^η, this is obvious.

Soundness:
We first need to establish that the probability for the malicious prover (who has no knowledge of the solution of the MinRank instance used and thus uses a bad witness) 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 if he commits a bad witness:

- He obtains the value $v = 0$ when executing the MPC protocol;

- The verifier believes that the value obtained v is 0.

We suppose here that the malicious prover commits a bad witness. Then, the first case occurs with probability $p = \frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}$ since it is the false positive rate of the protocol Π^η.

In the second case, the malicious prover needs to alter the communications in order to pass the verification. More precisely, he needs to alter the value of some share(s) of $\boldsymbol{\alpha}$ in every MPC protocol. Among the N leaf shares, only one 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 doesn't have a good witness $\boldsymbol{x}$. This means the malicious prover can only cheat on one share exactly. However, cheating on one share means cheating on one main share on all the D dimensions, as the main shares are the sum of leaves that have the same index i_j 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 party. This happens with probability $\frac{1}{N}$ (as the prover doesn't know the value of i^* before cheating).
This is the only pattern to cheat and avoid detection (as shown just now, we cannot cheat on more than one leaf), since cheating on 1 main party for each dimension is exactly equivalent to cheating on 1 leaf party.

Thus, since there is no other cheating pattern possible, the probability for the malicious prover to be authenticated is at most $p + (1-p) \cdot \frac{1}{N} = \frac{1}{N} + (1 - \frac{1}{N}) \cdot p$.

We then need to show the soundness property in the theorem:
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:

- $[\![\boldsymbol{x}]\!], [\![\boldsymbol{\beta}]\!], [\![\boldsymbol{a}]\!]$ 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 $[\![\boldsymbol{x}]\!], [\![\boldsymbol{\beta}]\!], [\![\boldsymbol{a}]\!]$ 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 we can build an extraction algorithm $\mathcal{E}$ that obtains a good witness.

$\boldsymbol{x}$ is called a good witness if it is a solution to the MinRank instance defined by public data, i.e. $\boldsymbol{E} = \boldsymbol{M}_0 + \sum_{i=1}^{k} \boldsymbol{M}_i x_i$ and $\mathrm{W}_R(\boldsymbol{E}) \leq r$. Let R_h the random variable associated to the randomness in initial commitment, and r_h is the value it takes. For that, we will use the Splitting Lemma as is done in [FJR22] and [AMGH$^+$22].

To get these two transcripts, the extraction algorithm $\mathcal{E}$ 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, $\mathcal{E}$ repeats the process L times (where the value of L is made explicit hereafter) until finding another accepted transcript, $\mathcal{T}_2$, for which the leaf challenge, i_2^*, is different than i_1^*;

- If such a transcript $\mathcal{T}_2$ is found, then $\mathcal{E}$ recovers the witness, otherwise, if no such transcript is found after L attempts, $\mathcal{E}$ returns to the first step and tries with another r_h.

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 (Lemma 1), 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 have a good randomness. Furthermore, we know that if the malicious prover uses a bad witness, his probability to cheat is bounded from above by ϵ. Since the probability of success is greater than ϵ, 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:

$$\Pr[\text{Succ}_{\tilde{\mathcal{P}}} \cap (i_1^* \neq i_2^*)|r_h \ good] = \Pr[\text{Succ}_{\tilde{\mathcal{P}}}|r_h \ good] - \Pr[\text{Succ}_{\tilde{\mathcal{P}}} \cap (i_1^* = i_2^*)|r_h \ good]$$

$$\geq \Pr[\text{Succ}_{\tilde{\mathcal{P}}}|r_h \ good] - \frac{1}{N}$$

$$\geq (1-\delta)\tilde{\epsilon} - \frac{1}{N}$$

$$\geq (1-\delta)\tilde{\epsilon} - \epsilon \qquad (\text{since } \epsilon \geq \frac{1}{N} \text{ trivially})$$

Now that we have this lower bound, we want to estimate the number of times one has to repeat the protocol to find $\mathcal{T}_2$. For that, we take the opposite probability, i.e, $1 - \Pr[\text{Succ}_{\tilde{\mathcal{P}}} \cap (i_1^* \neq i_2^*)|r_h \ good]$, 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:

$$\left(1 - \Pr[\text{Succ}_{\tilde{\mathcal{P}}} \cap (i_1^* \neq i_2^*)|r_h \ good]\right)^L < \frac{1}{2}$$

$$L \cdot \ln(1 - ((1-\delta)\tilde{\epsilon} - \epsilon)) < -\ln(2)$$

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

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 has 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 calls the extractor $\mathcal{E}$ has to make to $\tilde{\mathcal{P}}$. After L calls (to find $\mathcal{T}_2$), if r_h is good (which happens with probability δ), 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, $\Pr[\text{no } \mathcal{T}_2|\text{Succ}_{\tilde{\mathcal{P}}}] = \frac{\delta}{2} + (1-\delta) = 1 - \frac{\delta}{2}$. If that happens, then, $\tilde{\mathcal{P}}$ has to return to the first step, i.e, find $\mathcal{T}_1$ again. This means:

$$\mathbb{E}(\tilde{\mathcal{P}}) \leq 1 + \left((1 - \Pr[\text{Succ}_{\tilde{\mathcal{P}}}])\mathbb{E}(\tilde{\mathcal{P}})\right) + \Pr[\text{Succ}_{\tilde{\mathcal{P}}}]\left(L + (1 - \frac{\delta}{2})\mathbb{E}(\tilde{\mathcal{P}})\right)$$

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 $\mathcal{T}_2$.

31

Since $\Pr[\mathsf{Succ}_{\tilde{\mathcal{P}}}] = \tilde{\epsilon}$ (by assumption), we can replace, and simplify the expression. We find then:

$$\mathbb{E}(\tilde{\mathcal{P}}) \leq 1 + \left(\left(1 - \tilde{\epsilon} \cdot \mathbb{E}(\tilde{\mathcal{P}})\right) + \tilde{\epsilon} \cdot \left(L + (1 - \frac{\delta}{2}) \cdot \mathbb{E}(\tilde{\mathcal{P}})\right)\right)$$

$$\mathbb{E}(\tilde{\mathcal{P}}) \leq 1 + \mathbb{E}(\tilde{\mathcal{P}}) - \tilde{\epsilon} \cdot \mathbb{E}(\tilde{\mathcal{P}}) + \tilde{\epsilon} \cdot L + \tilde{\epsilon} \cdot \mathbb{E}(\tilde{\mathcal{P}}) - \tilde{\epsilon} \cdot \frac{\delta}{2} \cdot \mathbb{E}(\tilde{\mathcal{P}})$$

$$\tilde{\epsilon} \cdot \frac{\delta}{2} \cdot \mathbb{E}(\tilde{\mathcal{P}}) \leq 1 + \tilde{\epsilon} \cdot L$$

$$\mathbb{E}(\tilde{\mathcal{P}}) \leq \frac{2}{\tilde{\epsilon} \cdot \delta}\left(1 + \tilde{\epsilon} \cdot L\right) = \frac{2}{\tilde{\epsilon} \cdot \delta}\left(1 + \tilde{\epsilon} \cdot \frac{\ln(2)}{(1 - \delta)\tilde{\epsilon} - \epsilon}\right)$$

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

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

This means we found an upper bound on the number of calls the extractor $\mathcal{E}$ has to make to $\tilde{\mathcal{P}}$ before retrieving a good witness, in the case where the probability to cheat was higher than ϵ.

Honest-Verifier Zero-Knowledge:
Consider a simulator, described in Fig. 14, which produces the transcript responses (h_0, ch_1, h_1, ch_2, 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.
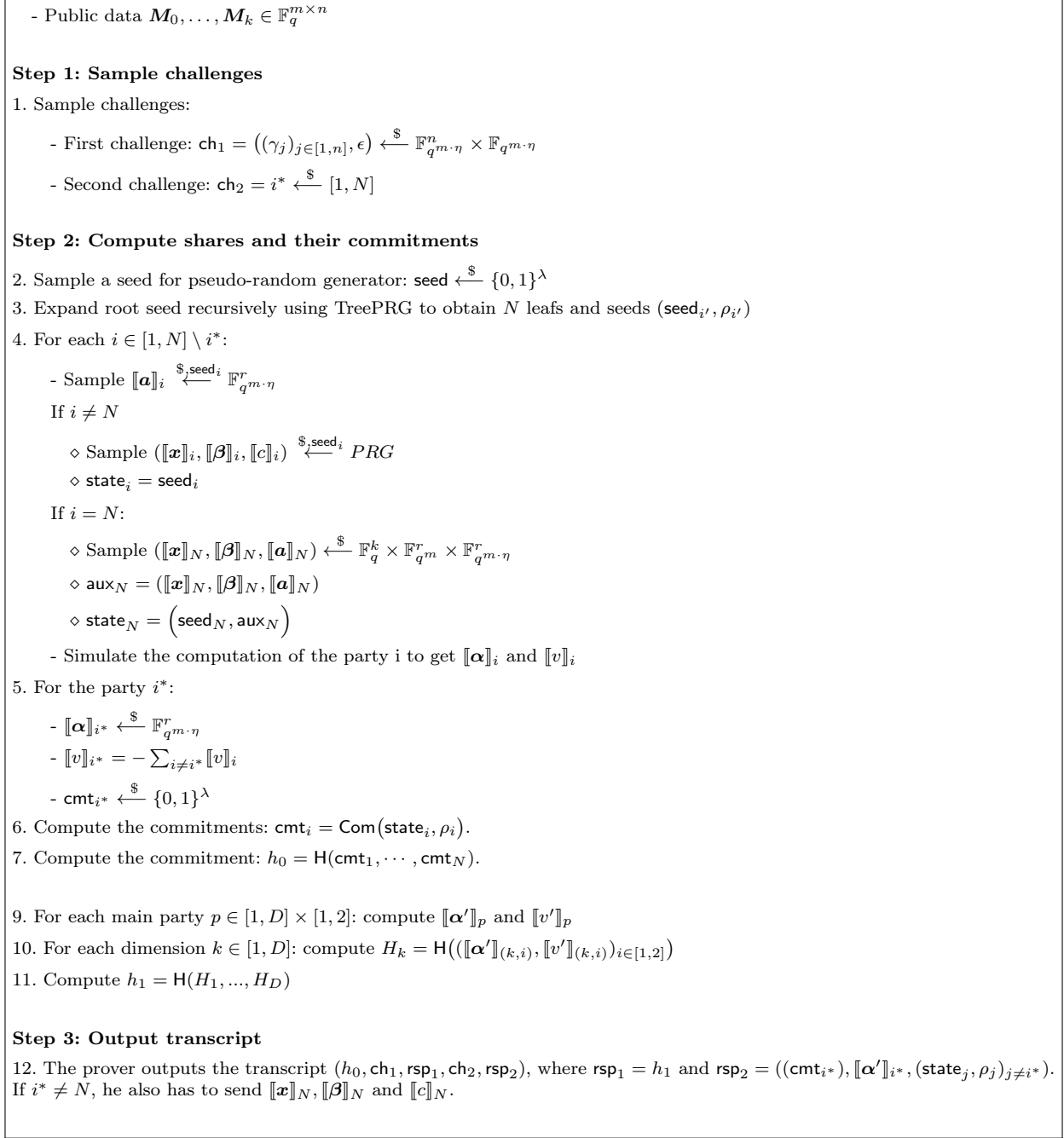
Fig. 14: HVZK simulator of the PoK with additive sharing and hypercube optimization

- **Simulator 0 (real world):** it takes in input the witness $\boldsymbol{x}$ and the challenges $\mathsf{ch}_1 = \big((\gamma_j)_{j \in [1,n]}, \epsilon\big)$ and $\mathsf{ch}_2 = i^*$. It worrectly executes the algorithm 3, hence its output is the correct distribution.

- **Simulator 1:** Same as the **Simulator 0**, but uses true randomness instead of seed-derived randomness for leaf i^*.
  If $i^* = N$, the leafs $[\![\boldsymbol{x}]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N$ are computed as in the MPC protocol.
  The pseudo-random generator is supposed to be (t, ϵ_{PRG})-secured, its outputs are indistinguish-

able 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 worlds, their distribution is also indistinguishable.

– **Simulator 2:** Replace the leafs $[\![\boldsymbol{x}]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N$ in **Simulator 1** by uniformly sampled values. Compute $[\![v]\!]_{i^*} = -\sum_{i \neq i^*} [\![v]\!]_i$. Note that this simulator becomes independent from the secret witness $\boldsymbol{x}$.
   If $i^* = N$, it only impacts the shares $[\![\boldsymbol{\alpha}]\!]_{i^*}$ and $[\![v]\!]_{i^*}$. Note that this change doesn't alter the uniform distribution of these values. It doesn't alter the distribution of any other leaf.
   If $i^* \neq N$, it only impacts $([\![\boldsymbol{x}]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N)$ in the simulated response and the values computed from them in the MPC protocol. It doesn't alter the distribution of other leaves. We observe that the shares in $([\![\boldsymbol{x}]\!]_N, [\![\boldsymbol{\beta}]\!]_N, [\![c]\!]_N)$ are calculated by adding a randomness value from each seed of party $i \neq i^*$, which correspond to adding a uniform random value from seed_{i^*}. Since this distribution was uniform 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, sample it uniformly from $\mathbb{F}_{q^{m \cdot \eta}}^r$. As in the previous simulator, it doesn't change their output distribution.

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

5.1.2 Security Proofs for the Threshold Sharing Protocol

Theorem 2. *The MinRank Proof of Knowledge protocol based on threshold secret sharing described in Fig.10 has the following properties:*

– **Completeness:** *A prover $\mathcal{P}$ who has the knowledge of a solution of a MinRank instance will always be accepted by the verifier.*

– **Soundness:** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that convinces the verifier to accept with probability*

$$\tilde{\epsilon} > \epsilon$$

with

$$\epsilon = \frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell \cdot (N - \ell)}{\ell + 1}$$

where ϵ is the soundness of the protocol in Fig.10, and p is the soundness of the MPC protocol used, i.e, $\frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}$.
Then, there is an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given a rewindable black-box access to $\tilde{\mathcal{P}}$, outputs either a solution of the MinRank instance, or a commitment collision by making a number of calls to $\tilde{\mathcal{P}}$ which is bounded by

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

– **Honest-Verifier Zero-Knowledge:** *If the pseudo-random generator algorithm PRG and the commitment* Com *are indistinguishable from the uniform random distribution, then the algorithm 10 is Honest-Verifier Zero Knowledge.*

Proof.

Completeness:

By construction, if the prover has knowledge of a solution of the MinRank instance, he will always be able to execute the protocol correctly, i.e, he will always obtain $\boldsymbol{\alpha}$ such that $v = 0$ when executing the MPC protocol Π^η, this is obvious.

Soundness:

The proof is rather long and complex. For this proof, we refer to [FR22], who proved this theorem for any MPC protocol and MPCitH protocol, as long as they lie in their model. Our threshold protocol is an exact application of their model. Hence, the proof of the above theorem is the same as the proof in appendix D of [FR22].

Honest-Verifier Zero-Knowledge:

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 done in [FR22, Appendix C]. Since we are in their model of MPCitH, the proof applies here as well.

5.2 Security proofs for the Signature Schemes

The proofs follow in large parts the proofs in [AMGH$^+$22], [FJR22] and [FR22].

5.2.1 Security Analysis of MIRA-Additive
We need to prove that the signature scheme is EUF-CMA secure:

Theorem 3. *Let the PRG used be (t, ϵ_{PRG})-secure, and ϵ_{MR} the advantage an adversary has over the MinRank problem. Consider $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4$ behave as random oracles, with an output of 2λ bits. Then, if an adversary makes q_i queries to H_i, q_S queries to the signing oracle, the probability for him to produce a forgery (EUF-CMA) for the MIRA Additive Signature Scheme (Fig.8) is:*

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

where τ is the number of rounds of the signature, $p = \frac{1}{q^{m\eta}} + \left(1 - \frac{1}{q^{m\eta}}\right)\frac{1}{q^{m\eta}}$, $X = \max_{i \in [0,q_2]}\{X_i\}$ 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. 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.
We note $\mathsf{Pr}_i[\mathsf{Forge}]$ the probability of forgery when considering game i. The aim of the proof is to find an upper bound on $\mathsf{Pr}_1[\mathsf{Forge}]$.

– **Game 1**
This is the interaction between $\mathcal{A}$ and the real signature scheme.
KeyGen generates $(\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k, \boldsymbol{x})$ and $\mathcal{A}$ receives $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k$. $\mathcal{A}$ can make queries to each H_i independently, and can make signing queries. At the end of the attack, $\mathcal{A}$ outputs a message/signature pair, (m, σ). The event Forge happens when σ is a valid signature of m and no signature of m has been queried 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 H_0, or H_1, or H_3, then, the forgery isn't valid.
The first step is to look at the number of times every H_i is called when calling the signing oracle. For H_0, we make $\tau \cdot N$ queries. The signing oracle contains also τ calls to H_1, one to H_2, $\tau \cdot D$ to H_3, and finally, a single one to H_4.
The number of queries to H_0 or H_1 or H_3 is then bounded from above by $q + \tau \cdot N \cdot q_S$, where q_i is the number of queries made by $\mathcal{A}$ to H_i, $q = \max\{q_0, q_1, q_2, q_3, q_4\}$ (we take q_2 and q_4 as well for q since we are giving an upper bound), and q_S is the number of queries to the signing oracle. 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):

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

– **Game 3**
The attacker now fails if the inputs to any of the H_i has already appeared in a previous query. Wlog. the adversary does not make such a query itself and, if it happens in a signing query, this means that (at least) the salt randomly sampled by the signing oracle appears in a previous hash or signing query. We can bound this event with:

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

– **Game 4**
When beginning the signature of the message m, h_1 and h_2 are sampled uniformly and expanded into $(\gamma_1^{(e)}, \ldots, \gamma_n^{(e)}, \epsilon^{(e)})_{e \in [1,\tau]}$, and $(i^{*(e)})_{e \in [1,\tau]}$. The game proceeds as before, but now we replace the queries to H_2 and H_4 by h_1 and h_2. If a query to H_2 or H_4 was already made, the attacker fails. However, this situation doesn't happen as **Game 3** would fail before. Hence,

$$\mathsf{Pr}_4[\mathsf{Forge}] = \mathsf{Pr}_3[\mathsf{Forge}]$$

– **Game 5**
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:

$$|\mathsf{Pr}_5[\mathsf{Forge}] - \mathsf{Pr}_4[\mathsf{Forge}]| \leq \epsilon_{PRG}$$

– **Game 6**
Finally, we say that an execution e^* of a query $h_2 = \mathsf{H}_4(m, \mathsf{salt}, h_1, [\![H_1^{(e)}]\!] \ldots [\![H_D^{(e)}]\!]_{e \in [1,\tau]})$ defines

a good witness $\boldsymbol{x}$ if:

- Each of the $H_k^{(e)}$ are the output of a query to H_3
- h_1 is the output of a query to H_2, 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 H_1, i.e,

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

-Each $\mathsf{cmt}_i^{(e)}$ is the output of a query to 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, $\boldsymbol{E} = \boldsymbol{M}_0 + \sum_{i=1}^k \boldsymbol{M}_i x_i$ such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$.
In case where such an execution happens, one can retrieve the correct witness from the states $\{\mathsf{state}_i\}_{i \in [1,N]}$ and, as a consequence, one can solve the MinRank instance. This means that $\mathsf{Pr}_6[\mathsf{Solve}] \leq \epsilon_{MR}$.

Finally, we only need to look at the upper bound of $\left|\mathsf{Pr}_6[\mathsf{Forge} \cap \overline{\mathsf{Solve}}]\right|$. 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:

Solve doesn't happen here, meaning that, to have a forgery after a query to 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 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 $\mathsf{Pr}[e \in G_2(Q_2, h_1) \mid \overline{\mathsf{Solve}}] \leq p = \frac{1}{q^{mn}} + \left(1 - \frac{1}{q^{mn}}\right)\frac{1}{q^{mn}}$. 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 $(Q_{2\mathrm{best}}, h_{1\mathrm{best}})$ such that $\#G_2(Q_2, h_1)$ is maximized, i.e,

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

Cheating at the second round. Now, we need to look at the cheating in the second round, i.e, the queries to 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 leaf (it is already established that is isn't possible to cheat on less, or on more, than one leaf). He only needs to cheat when the value of $h_1^{(e)}$ is wrong, i.e, he needs to cheat for every round $e \notin G_2(Q_{2\mathrm{best}}, h_{1\mathrm{best}})$. 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 $\left(\frac{1}{N}\right)^{\tau - \#G_2(Q_{2\mathrm{best}}, h_{1\mathrm{best}})}$

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 τ_1 possible, we have then the upper bound:

$$\text{Pr}_6[\text{Forge} \cap \overline{\text{Solve}}] \leq \text{Pr}[X + Y = \tau]$$

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   Security Analysis of MIRA-Threshold

Theorem 4. *Let the PRG used be (t, ϵ_{PRG})-secure, and ϵ_{MR} the advantage an adversary has over the MinRank problem. Consider $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$ and H_M behave as random oracles, with an output of 2λ bits (H_M is the function used for the Merkle Tree). Then, if an adversary makes q_i queries to H_i, q_S queries to the signing oracle, q_M queries to H_M, the probability for him to produce a forgery (EUF-CMA) for the MIRA Threshold Signature Scheme (Fig.12) is:*

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

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

Proof. In this proof, we will adopt a game hopping strategy in order to find the upper bound.
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.
We note $\text{Pr}_i[\text{Forge}]$ the probability of forgery when considering game i. The aim of the proof is to find an upper bound on $\text{Pr}_1[\text{Forge}]$.

– **Game 1**
This is the interaction between $\mathcal{A}$ and the real signature scheme.
KeyGen generates $(\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k, \boldsymbol{x})$ and $\mathcal{A}$ receives $\boldsymbol{M}_0, \ldots, \boldsymbol{M}_k$. $\mathcal{A}$ can make queries to each H_i independently, and can make signing queries. At the end of the attack, $\mathcal{A}$ outputs a message/signature pair, (m, σ). The event 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 H_0 or on H_M, then the forgery isn't valid. Here, H_0 is called q_0 times by $\mathcal{A}$, H_M q_M times. When $\mathcal{A}$ calls the signing oracle, there are in total: $\tau \cdot N$ calls to H_0, $\tau \cdot (2 \cdot N - 1)$ calls to H_M, and one to H_1 and H_2. In this game, only H_0 and H_M are of interest, but for a simpler notation, we

will take $q = \max\{q_0, q_1, q_2, q_M\}$ (as it is an upper bound we are looking for, this is fine). We can then give an upper bound to the queries made by $\mathcal{A}$ to the hash functions, which is then: $q + \tau \cdot (2 \cdot N - 1) \cdot q_S$ 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

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

– **Game 3**
The attacker now fails if the inputs to any of the H_i has already appeared in a previous query. Wlog. the adversary does not make such a query itself and, if it happens in a signing query, this means that (at least) the salt randomly sampled by the signing oracle appears in a previous hash or signing query. Since we don't use salt in the Merkle Tree, this will only concern $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$. We sample salt q_S time (once by signing oracle query), and $3 \cdot q$ times as well (each time we call $\mathsf{H}_0, \mathsf{H}_1$ or H_2). If there is an input which already appears for H_M, this must be because a collision has been found either on H_M or on H_0. However, we already excluded this in **Game 2**. This means we can give the following bound:

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

– **Game 4**
When beginning the signature of the message m, h_1 and h_2 are sampled uniformly and expanded into $\gamma_1^{(e)}, \ldots, \gamma_n^{(e)}, \epsilon^{(e)}$, and $i^{*(e)}$. The game proceeds as before, but now we replace the queries to H_2 and H_4 by h_1 and h_2. If a query to H_2 or H_4 was already made, the attacker fails. However, this situation doesn't happen as **Game 3** would fail before. Hence,

$$\mathsf{Pr}_4[\mathsf{Forge}] = \mathsf{Pr}_3[\mathsf{Forge}]$$

– **Game 5**
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:

$$|\mathsf{Pr}_5[\mathsf{Forge}] - \mathsf{Pr}_4[\mathsf{Forge}]| \leq \epsilon_{PRG}$$

– **Game 6**
Finally, we say that an execution e^* of a query $h_2 = \mathsf{H}_2(m, \mathsf{pk}, \mathsf{salt}, h_1, ([\![\boldsymbol{\alpha}^{(e)}]\!]_i, [\![v^{(e)}]\!]_i)_{i \in S, e \in [1, \tau]})$ defines a good witness $\boldsymbol{x}$ if:

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

$$h_1 = \mathsf{H}_1(\mathsf{salt}, m, h_0^{(1)}, \ldots, h_0^{(\tau)})$$

- 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 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, $\boldsymbol{E} = \boldsymbol{M}_0 + \sum_{i=1}^k \boldsymbol{M}_i x_i$ such that $\mathrm{W}_R(\boldsymbol{E}) \leq r$.
In case where such an execution happens, one can retrieve the correct witness from the states $\{\mathsf{state}_i\}_{i \in [1,N]}$ and, as a consequence, one can solve the MinRank instance. This means that $\mathsf{Pr}_6[\mathsf{Solve}] \leq \epsilon_{MR}$.

Finally, we only need to look at the upper bound of $\left| \mathsf{Pr}_6[\mathsf{Forge} \cap \overline{\mathsf{Solve}}] \right|$. This probability is upper bounded by the value

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

with $X = \max_{i \in [0,q_1]}\{X_i\}$ with $X_i \sim \mathcal{B}(\tau, \binom{N}{\ell+1} \cdot p)$, $Y = \max_{i \in [0,q_2]}\{Y_i\}$ 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] proposes an attack more efficient than brute force 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 attack which consists in guessing separately the two challenges of the protocol. It results an additive cost rather than the expected multiplicative cost. The cost to forge a valid transcript for a 5-round proof of knowledge corresponds to the cost of the optimal trade-off between the work needed to pass the first step and the work needed to pass the second step. To run the attack, one can find the optimal number of repetitions for the brute-force work of the first step with the formula:

$$\tau' = \arg \min_{0 \leq \tau' \leq \tau} \left\{ \frac{1}{\sum_{i=\tau'}^{\tau} \binom{\tau}{i} P_1^i (1 - P_1)^{\tau - i}} + \left(\frac{1}{P_2}\right)^{\tau - \tau'} \right\}$$

where P_1 and P_2 are the probabilities to pass respectively the first and the second challenges for one repetition.

6.1.1 Cost of forgery of MIRA-Additive

In the additive case, one obtains:

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

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

6.1.2 Cost of forgery of MIRA-Threshold

In the threshold case, one obtains:

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

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

6.2 Attacks against MinRank

In this section, we briefly describe the most effective attacks on MinRank. A reader can refer to [BB22], [GND23], [BBB$^+$22], [BBC$^+$20], and [GC00] for more details on the attacks.

To begin with, the following hybrid approach can benefit to all the other attacks.

Hybrid approach

[BBB$^+$22] introduced a generic approach to improve all the attacks on MinRank. The idea of the attack is to solve smaller instances of MinRank instead. The complexity is given by

$$\min_a \left(q^{ar} \mathbb{C}_{\mathcal{A}}(q, m, n - a, K - am, r) \right) \tag{5}$$

where $\mathbb{C}_{\mathcal{A}}(q, m, n, K, r)$ is the cost of an algorithm $\mathcal{A}$ to solve a MinRank instance.

6.2.1 The Kernel Attack

The kernel attack was described by Goubin and Courtois in [GC00]. The idea of the attack is to take random vectors, and hoping that they are in the kernel of $\boldsymbol{E}$. Since $\boldsymbol{E}$ is of size $m \times n$, and is of rank at most r, $\text{Ker}(\boldsymbol{E})$ will be a matrix of dimensions $n \times (n - r)$ at least. Thus, if $v \xleftarrow{\$} \mathbb{F}_q^n$, $\Pr[v \in \text{Ker}(\boldsymbol{E})] = \frac{q^{n-r}}{q^n} = \frac{1}{q^r}$. Then, if we get l independant vectors in $\text{Ker}(\boldsymbol{E})$, and set the matrix $\boldsymbol{X}$ whose columns are the l vectors, we can compute $(\boldsymbol{M}_0 + \sum_{i=1}^{k} x_i \boldsymbol{M}_i) X$, which gives us a linear system in $x_1 \ldots x_k$, and $m \cdot l$ equations (since we have $(\boldsymbol{M}_0 + \sum_{i=1}^{k} x_i \boldsymbol{M}_i) X = 0$). Thus with $l = \lceil \frac{k}{m} \rceil$, we have a unique solution to the system, which we can find with linear algebra.
As to the complexity of the attack, it is quite obvious that it is in $O(q^{r \lceil \frac{k}{m} \rceil})$ to find the vectors in the kernel, and in $O(k^\omega)$ to solve the linear system. Hence, the total complexity is

$$O(q^{r \lceil \frac{k}{m} \rceil} k^\omega)$$

6.2.2 Algebraic Attacks

Minors Modeling

The modeling was introduced and studied in [FSS10] and [FSS13]. This modeling uses the minors of the matrix $\boldsymbol{E}$, where the x_i are still unknowns. It was also improved in [GND23]. We refer to these papers for the complexity of the attack.

Support Minors Modeling

The Support Minors modeling was introduced in [BBC$^+$20]. This idea also uses minors of a matrix, giving us an other system of equations. With this approach, the complexity is of

$$\mathcal{O}\left(N_b M_b{}^{\omega-1}\right),$$

where

$$N_b = \sum_{i=1}^{b} (-1)^{i+1} \binom{n}{r+i} \binom{k+b-1-i}{b-i} \binom{m+i-1}{i}. \tag{6}$$

$$M_b = \binom{k+b-1}{b} \binom{n}{r}. \tag{7}$$

and b is the degree to which we augment the Macaulay matrix of the system ([BBB$^+$22]).

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. Cryptology ePrint Archive, Paper 2002/022, 2002. https://eprint.iacr.org/2002/022.

AHIV17.   Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2087–2104, New York, NY, USA, 2017. Association for Computing Machinery.

AMGH$^+$22. Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The Return of the SDitH. Cryptology ePrint Archive, Paper 2022/1645, 2022. https://eprint.iacr.org/2022/1645.

ARZV22.   Gora Adj, Luis Rivera-Zamarripa, and Javier Verbel. MinRank in the Head: Short Signatures from Zero-Knowledge Proofs. Cryptology ePrint Archive, Paper 2022/1501, 2022. https://eprint.iacr.org/2022/1501.

BB22.   Magali Bardet and Manon Bertin. Improvement of Algebraic Attacks for Solving Superdetermined MinRank Instances. In Jung Hee Cheon and Thomas Johansson, editors, *pqcrypto 2022*, volume 13512 of *lncs*, pages 107–123, Cham, September 2022. 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. Cryptology ePrint Archive, Paper 2022/1031, 2022. https://eprint.iacr.org/2022/1031.

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*, pages 507–536. Springer International Publishing, 2020.

BESV22.   Emanuele Bellini, Andre Esser, Carlo Sanna, and Javier Verbel. MR-DSS – Smaller MinRank-based (Ring-)Signatures. Cryptology ePrint Archive, Paper 2022/973, 2022. https://eprint.iacr.org/2022/973.

BFS99.   Jonathan F Buss, Gudmund S Frandsen, and Jeffrey O Shallit. The Computational Complexity of Some Problems of Linear Algebra. *Journal of Computer and System Sciences*, 58(3):572–596, 1999.

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.

Cou01.   Nicolas Courtois. La sécurité des primitives cryptographiques basées sur des problèmes algébriques multivariables MQ, IP, MinRank, HFE, 2001.

Fen22.   Thibauld Feneuil. Building MPCitH-based Signatures from MQ, MinRank, Rank SD and PKP. Cryptology ePrint Archive, Paper 2022/1512, 2022. https://eprint.iacr.org/2022/1512.

FJR22.   Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs. Cryptology ePrint Archive, Paper 2022/188, 2022. https://eprint.iacr.org/2022/188.

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. https://eprint.iacr.org/2022/1407.

FSS10.   Jean-Charles Faugère, Mohab Safey El Din, and Pierre-Jean Spaenlehauer. Computing loci of rank defects of linear matrices using Gröbner bases and applications to cryptology. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 2010, Munich, Germany, July 25-28, 2010*, pages 257–264, 2010.

FSS13.   Jean-Charles Faugère, Mohab Safey El Din, and Pierre-Jean Spaenlehauer. On the complexity of the generalized minrank problem. *JSC*, 55:30–58, 2013.

GC00.   Louis Goubin and Nicolas T. Courtois. Cryptanalysis of the TTM Cryptosystem. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2000.

GND23.   Sriram Gopalakrishnan, Vincent Neiger, and Mohab Safey El Din. Refined f_5 algorithms for ideals of minors of square matrices, 2023.

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

KZ20.   Daniel Kales and Greg Zaverucha. An Attack on Some Signature Schemes Constructed From Five-Pass Identification Schemes. Cryptology ePrint Archive, Paper 2020/837, 2020. https://eprint.iacr.org/2020/837.

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. *J. Cryptology*, 13:361–396, 2000.