MEDS
Matrix Equivalence Digital Signature

https://www.meds-pqc.org/

Tung Chou, Ruben Niederhagen, Edoardo Persichetti, Lars Ran,
Tovohery Hajatiana Randrianarisoa, Krijn Reijnders, Simona Samardjiska,
and Monika Trimoska

Academia Sinica, Taipei, Taiwan
University of Southern Denmark, Odense, Denmark
Florida Atlantic University, Boca Raton, USA
Sapienza University, Rome, Italy
Umeå University, Umeå, Sweden
Radboud Universiteit, Nijmegen, The Netherlands

Submitters: The team listed above is joint principal submitter. There are no auxiliary submitters.

Inventors/Developers: Submitters. Relevant prior work is credited where appropriate.

Owners: Submitters.

Email Address: author-contact@meds-pqc.org

Postal Address, Telephone: Simona Samardjiska, Digital Security Group, Radboud University, Postbus 9010, 6500 GL Nijmegen, The Netherlands, +31 24-3653456.

Signature: [illegible] See also "NIST Submission Statement" by each submitter.

Version: 1.0.1 **Date:** June 30, 2023

Contents

1 Introduction

The Matrix Equivalence Digital Signature (MEDS) scheme is a digital signature scheme based on the difficulty of finding an isometry between two equivalent matrix rank-metric codes. From the rank-equivalence problem, we obtain a zero-knowledge identification scheme using multiple rounds of a Sigma protocol. The signature scheme is then obtained via the Fiat-Shamir transform. As described in this document, various optimizations are incorporated into the scheme, which allows for multiple tradeoffs. This scheme has been introduced in the publication [CNP+23]:

> Tung Chou, Ruben Niederhagen, Edoardo Persichetti, Tovohery Hajatiana Randrianarisoa, Krijn Reijnders, Simona Samardjiska, and Monika Trimoska: *"Take your MEDS: Digital Signatures from Matrix Code Equivalence"*. Progress in Cryptology — AfricaCrypt 2023. Lecture Notes in Computer Science, Springer, 2023.

2 Tools

In this section we introduce the basic tools that we need to describe MEDS. More detailed mathematical background will be introduced in the rest of the document.

2.1 Notation and Main Definitions

Notation and parameters of the MEDS scheme are shown in Table 1.

Matrices. We denote matrices with bold capital letters, e.g., $\mathbf{M}$, $\mathbf{A}$, and $\mathbf{B}$.

Submatrices. We denote submatrices with square brackets, e.g., $\mathbf{M}[a, b; c, d]$ denotes the submatrix of the intersection of rows a to b and columns c to d of matrix $\mathbf{M}$. If no row (column) range is provided, all rows (columns) are included. The element in row i and column j of a matrix $\mathbf{M}$ is denoted as $\mathbf{M}[i; j]$.

Matrix Rank Metric Code. An $[m \times n, k]$ matrix rank metric code over $\mathbb{F}_q$ is a k-dimensional subspace of $\mathbb{F}_q^{m\times n}$.

Isometry. An isometry on $\mathbb{F}_q^{m\times n}$ is an $\mathbb{F}_q$-linear map ϕ defined by two matrices $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$ such that $\phi(\mathbf{M}) = \mathbf{AMB}$ for all $\mathbf{M} \in \mathbb{F}_q^{m\times n}$.[1]

Matrix Code Equivalence Problem. The MCE Problem asks whether there exists an isometry between two matrix rank metric codes. Its computational form is written as follows.

- **Input**: Two $[m \times n, k]$ rank metric codes $\mathcal{C}$ and $\mathcal{D}$.
- **Problem**: Find (if any) an isometry ϕ on $\mathbb{F}_q^{m\times n}$ such that $\mathcal{C} = \phi(\mathcal{D})$.

2.2 Signatures from Matrix Equivalence

Our scheme is based on a three-pass Sigma protocol, which we present in Figure 1.

[1]This definition of isometries restricts to maps on the whole matrix space, i.e. $\phi : \mathbb{F}_q^{m\times n} \to \mathbb{F}_q^{m\times n}$, compared to the usual definition of an isometry specifically between two $[m \times n, k]$ matrix rank codes $\mathcal{C}$, $\mathcal{D}$, i.e. $\phi : \mathcal{C} \to \mathcal{D}$.

Scheme Parameters:	
q	Size of the base field, a prime.
m, n	Codeword sizes, two integers.
k	Code dimension, an integer.
λ	Security parameter, an integer.
Protocol Objects:	
$\mathbb{F}_q$	Finite fields with q elements.
$\mathrm{GL}_n(q)$	Group of invertible matrices of size n and elements in $\mathbb{F}_q$.
$\mathbf{G}_0, \mathbf{G}_1$	Public matrices in $\mathbb{F}_q^{k \times mn}$.
$\mathbf{A} \in \mathrm{GL}_m(q), \mathbf{B} \in \mathrm{GL}_n(q)$	Secret invertible matrices.
$\tilde{\mathbf{A}} \in \mathrm{GL}_m(q), \tilde{\mathbf{B}} \in \mathrm{GL}_n(q)$	Ephemeral invertible matrices.
H	Hash function, with domain/range as specified.
$\mathsf{SF}(\mathbf{A})$	Reduced-row echelon form of a matrix $\mathbf{A}$.
$\mathbf{A}^\top$	Transpose of a matrix $\mathbf{A}$.
$\mathbf{A} \otimes \mathbf{B}$	Kroenecker product between two matrices $\mathbf{A}$ and $\mathbf{B}$.
$\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G})$	Simplified notation for the operation $\mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B})$.
$\mathbf{I}_d$	Identity matrix of size d.
$\mathbf{U}_d$	Upper shift matrix of size d (identity shifted by one column to the right).
$\mathbf{0}_{d_1 \times d_2}$	Zero matrix of size $d_1 \times d_2$.
cmt	Commitment, a hash digest.
ch	Challenge, a bit.
rsp	Response, a tuple in $\mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.

Table 1: Notation and parameters of the MEDS scheme.

Public Data
$q, m, n, k, \lambda \in \mathbb{N}$.
$\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$.

I. Keygen()

1. $\mathbf{G}_0 \xleftarrow{\$} \mathbb{F}_q^{k \times mn}$.
2. $\mathbf{A}, \mathbf{B} \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
3. Compute $\mathbf{G}_1 = \mathsf{SF}(\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G}_0))$.
4. Set $\mathsf{sk} = (\mathbf{A}, \mathbf{B})$ and $\mathsf{pk} = (\mathbf{G}_0, \mathbf{G}_1)$.

II. Commit(pk)

1. $\tilde{\mathbf{A}}, \tilde{\mathbf{B}} \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
2. Compute $\tilde{\mathbf{G}} = \mathsf{SF}(\pi_{\tilde{\mathbf{A}},\tilde{\mathbf{B}}}(\mathbf{G}_0))$.
3. Compute $h = \mathsf{H}(\tilde{\mathbf{G}})$.
4. Set $\mathsf{cmt} = h$.
5. Send cmt to verifier.

III. Challenge()

1. $c \xleftarrow{\$} \{0, 1\}$.
2. Set $\mathsf{ch} = c$.
3. Send ch to prover.

IV. Response(sk, pk, cmt, ch)

1. If $\mathsf{ch} = 0$ set $(\mu, \nu) = (\tilde{\mathbf{A}}, \tilde{\mathbf{B}})$.
2. If $\mathsf{ch} = 1$ set $(\mu, \nu) = (\tilde{\mathbf{A}}\mathbf{A}^{-1}, \mathbf{B}^{-1}\tilde{\mathbf{B}})$.
3. Set $\mathsf{rsp} = (\mu, \nu)$.
4. Send rsp to verifier.

V. Verify(pk, cmt, ch, rsp)

1. If $\mathsf{ch} = 0$ compute
   $h' = \mathsf{H}(\mathsf{SF}(\pi_{\mu,\nu}(\mathbf{G}_0)))$.
2. If $\mathsf{ch} = 1$ compute
   $h' = \mathsf{H}(\mathsf{SF}(\pi_{\mu,\nu}(\mathbf{G}_1)))$.
3. Accept if $h' = \mathsf{cmt}$ or reject otherwise.

Figure 1: MEDS Sigma Protocol

3 Protocol Description (2.B.1)

3.1 Design Rationale

What is the Security of MEDS? The mathematical problem behind MEDS is the Matrix Code Equivalence (MCE) problem. This problem is considered to be hard after extensive study, and the best algorithm to solve it is exponential, when the parameters are carefully chosen. Such hardness is at the basis of the security of the zero-knowledge scheme and when applying the Fiat-Shamir transform [FS87], we obtain Existential Unforgeability against Chosen Message Attacks (EUF-CMA). The transformation operates in the Random Oracle Model (ROM), and plausibly provides security in the Quantum Random Oracle Model (QROM) as well, as argued for instance in [DFMS19, LZ19]. The security of the MCE Problem is discussed in Section 4.

What is the Matrix Code Equivalence Problem? The MCE problem is exactly the vectorization problem for the group action of isometries on matrix rank-metric codes. For a matrix $\mathbf{M} \in \mathbb{F}_q^{m\times n}$, one can concatenate its rows to obtain a vector in $\mathbb{F}_q^{mn}$. This gives an isomorphism between $\mathbb{F}_q^{m\times n}$ and $\mathbb{F}_q^{mn}$ and it allows to represent a matrix rank metric code $\mathcal{C}$ as a k-dimensional subspace of $\mathbb{F}_q^{mn}$. In this setting, the isomorphism of $\mathbb{F}_q^{m\times n}$ defined by $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$ can be expressed by a multiplication on the right by the Kroenecker product $\mathbf{A}^\top \otimes \mathbf{B}$. If, in addition, we represent the two codes $\mathcal{C}$ and $\mathcal{D}$ by their generator matrices $\mathbf{G}_1$ and $\mathbf{G}_2$, in standard form, then an isometry ϕ such that $\phi(\mathcal{C}) = \mathcal{D}$ can be expressed by a mapping between the generator matrices where $\mathbf{G}_1 \mapsto \mathbf{T}\mathbf{G}_1(\mathbf{A}^\top \otimes \mathbf{B})$ for some matrix $\mathbf{T} \in \mathrm{GL}_k(q)$. This representation allows an explicit formulation of the MCE problem given in Section 2.1.

3.2 Protocol Steps

Iterating. The identification protocol in Figure 1 only provides *soundness* $1/2$, meaning that a malicious party can successfully impersonate an honest prover half of the times. To achieve the authentication level required, then, it is necessary to iterate the protocol t times, where in the simplest case $t = \lambda$. By "iterate" here we intend repeating the Commit, Challenge and Response phases independently; the verifier will verify each response separately and only accept if verification is passed in all iterations.

Fiat-Shamir. The Fiat-Shamir transformation [FS87] is applied to the iterated protocol in order to obtain a signature scheme. Informally, this transformation replaces the role of the verifier in the challenge step by producing the string of challenges via a collision-resistant hash function, which is computed on the message to be signed, together with the commitments for each round; in doing so, it turns the protocol from interactive to non-interactive. Note that, if the scheme is designed to be *commitment-recoverable* (as is the case for MEDS), it is not necessary to transmit the commitments as part of the signature; this can instead include the hash digest, which can then be used to verify correctness of the signatures once commitments are, as the name says, recovered on the verifier side.

Multiple Public Keys. It is possible to greatly reduce the soundness error by expanding the public key [DG19]. In our case, this means including multiple generator matrices of the form $\mathbf{G}_i = \mathsf{SF}(\mathbf{G}_0(\mathbf{A}_i^\top \otimes \mathbf{B}_i))$, corresponding to as many secret matrix pairs $(\mathbf{A}_i, \mathbf{B}_i)$, for $1 \le i \le s$. The value s, an integer, becomes then a system parameter. The verifier's challenge then asks to provide an isometry starting from a specific key $\mathbf{G}_i$. Since the challenge space is larger, fewer rounds are necessary to achieve the same authentication level, which allows to reduce the signature size (at the cost of increasing in the public key).

Fixed-Weight Challenge Strings. When the isometry queried is the one which comes from $\mathbf{G}_0$, the response consists of matrices $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$, which are generated uniformly at random; it is then sufficient to transmit only the seed used to generate them. In other words, the challenge corresponding to 0 is much "lighter" than the others, and so the communication cost can be improved by adjusting the probability distribution of the challenge string, to make this possibility more likely to happen [BKP20a]. This means that individual rounds require less communication on average (at the cost of increasing the number of rounds).

Seed Tree. To efficiently represent the t seeds used in the signing and verification algorithms of MEDS, we use a binary "seed tree" [BKP20a]. To begin, the root of the tree is set by a randomly chosen master seed. For every node, we generate its two children by feeding a PRNG with the node value and parse the PRNG output as its two children. This procedure is iterated for $\lceil \log_2(t) \rceil$ times, so that we end up with a layer having $2^{\lceil \log_2(t) \rceil} \geq t$ seeds. When one needs to communicate all but a subset of the t seeds, say, for instance, all except those indexed by a set $J \subset \{0, \cdots, t-1\}$ of size w, it is possible to exploit the tree structure to reduce the number of bits transmitted. The idea to improve efficiency is that of sending parent nodes, whenever possible: the verifier will repeat the procedure to generate the children nodes, and will thus obtain the required seeds, while minimizing the amount of space required in the signature. In the worst case, communicating the $t - w$ seeds requires the following amount of bits:

$$\lambda \left(2^{\lceil \log_2(w) \rceil} + w(\lceil \log_2(t) \rceil - \lceil \log_2(w) \rceil - 1)\right).$$

Public Key Compression. To combat the trade-off of public key vs. signature size, we introduced a technique for public key compression in [CNP+23] that trades public key size for private key size inspired by the trade-off in the key generation of Rainbow [DCP+20] and UOV [BCH+23]. Instead of generating the secret $(\mathbf{A}_i, \mathbf{B}_i)$ from a secret seed and then deriving the public $\mathbf{G}_i$, we generate $\mathbf{G}_i$ partially from a public seed and then use it to find $(\mathbf{A}_i, \mathbf{B}_i)$ and the rest of the public key $\mathbf{G}_i$. We start by performing a secret change of basis of $\mathbf{G}_0$ by multiplying it by a secret matrix $\mathbf{T}_i \in \mathrm{GL}_k(q)$ generated from a secret seed to obtain $\mathbf{G}'_0$. From a public seed we generate a complete $m \times n$ codeword $\mathbf{P}_0^{(i)}$ and the top $m-1$ rows of a codeword $\mathbf{P}_1^{(i)}$. Using these and the codewords $\mathbf{P}_0^{(0)}, \mathbf{P}_1^{(0)}$ corresponding to the first two rows from $\mathbf{G}'_0$ we find $\mathbf{A}_i$ and $\mathbf{B}_i$ from the linear relations: $\mathbf{P}_0^{(i)}\mathbf{B}_i^{-1} = \mathbf{A}_i\mathbf{P}_0^{(0)}$ and $\mathbf{P}_1^{(i)}\mathbf{B}_i^{-1} = \mathbf{A}_i\mathbf{P}_1^{(0)}$. To obtain a single solution we fix one value of $\mathbf{A}_i$ (this is equivalent to considering the isometry projectively). After the isometry $(\mathbf{A}_i, \mathbf{B}_i)$ is found, we proceed to finding the rest of the matrices $\mathbf{P}_j^{(i)} = \mathbf{A}_i\mathbf{P}_j^{(0)}\mathbf{B}_i$ for all $j \in \{3, \ldots, k\}$ and then construct the public $\mathbf{G}_i$.

This step during key generation reduces the space required for the public matrices by $2s(mn - k)$ and can also lead to reduction of the signature size by appropriate balancing. On the other hand it introduces computational overhead. We take care of this by introducing an optimization – instead of generating $\mathbf{P}_0^{(i)}$ and $\mathbf{P}_1^{(i)}$ randomly, we take these to be the identity matrix, and the identity shifted by one position to the right, respectively, which makes the resulting system sparse and structured (see Section 5.2, paragraph "System solving", for some implementation notes).

3.3 Supporting Functions

In the description of the MEDS signature scheme, we use the following definitions and subroutines:

$\mathcal{B}$ denotes one byte, i.e., $\mathcal{B} = \{0, \dots, 255\}$.

$\mathsf{F_q} : \mathbb{Z} \mapsto \mathbb{F}_q$ maps an integer a with $0 \leq a < q$ to the corresponding field element.

$\mathsf{Z} : \mathbb{F}_q \mapsto \mathbb{Z}$ maps a field element to the corresponding integer.

$\mathsf{BitLen} : \mathbb{Z} \mapsto \mathbb{Z}$ returns the number of bits required to store an integer value, i.e. $\mathsf{BitLen}(i) = \lceil \log_2(i) \rceil$.

$\mathsf{ByteLen} : \mathbb{Z} \mapsto \mathbb{Z}$ returns the number of bytes required to store a integer value, i.e., $\mathsf{ByteLen}(i) = \lceil \log_2(i)/8 \rceil$.

$\mathsf{Randombytes} : \mathbb{Z} \mapsto \mathcal{B}^*$ takes an integer l as input and returns l random bytes $b_0, \dots, b_{l-1} \in \mathcal{B}$.

$\mathsf{XOF} : \mathcal{B}^* \times \mathbb{Z} \times \cdots \times \mathbb{Z} \mapsto \mathcal{B}^* \times \cdots \times \mathcal{B}^*$ is an extendable output function that takes a seed and a sequence of length values as inputs and produces a sequence of pseudo-random byte vectors of lengths according to the input length sequence: $\mathsf{XOF}(\sigma, \ell_0, \dots, \ell_i) \mapsto \mathcal{B}^{\ell_0} \times \cdots \times \mathcal{B}^{\ell_i}$. The call $\mathsf{XOF}(\sigma, *)$ produces a byte stream.

$\mathsf{RowsToMatrices} : \mathbb{F}_q^{k \times mn} \mapsto \{\mathbb{F}_q^{m \times n}\}^k$ takes a matrix $\mathbf{M} \in \mathbb{F}_q^{k \times mn}$ as input, maps each row $i \in \{0, \dots, k-1\}$ of $\mathbf{M}$ to a matrix $\mathbf{P}_i \in \mathbb{F}_q^{m \times n}$ such that $\mathbf{P}_i[\lfloor j/n \rfloor; j \bmod n] = \mathbf{M}[i, j]$ for $j \in \{0, \dots, mn-1\}$, and returns k matrices $\mathbf{P}_0, \dots, \mathbf{P}_{k-1} \in \mathbb{F}_q^{m \times n}$.

$\mathsf{MatricesToRows} : \{\mathbb{F}_q^{m \times n}\}^k \mapsto \mathbb{F}_q^{k \times mn}$ takes k matrices $\mathbf{P}_0, \dots, \mathbf{P}_{k-1}$ as input, generates a matrix $\mathbf{M} \in \mathbb{F}_q^{k \times mn}$ by mapping $\mathbf{P}_i$, $i \in \{0, \dots, k-1\}$, to row i of $\mathbf{M}$ such that $\mathbf{M}[i, j] = \mathbf{P}_i[\lfloor j/n \rfloor; j \bmod n]$ for $j \in \{0, \dots, mn-1\}$, and returns $\mathbf{M} \in \mathbb{F}_q^{k \times mn}$.

$\mathsf{SF} : \mathbb{F}_q^{a \times b} \mapsto \mathbb{F}_q^{a \times b} \cup \{\perp\}; a, b \in \mathbb{Z}$ returns the systematic form of a matrix $\mathbf{M} \in \mathbb{F}_q^{a \times b}$ or $\perp$ if the systematic form does not exist.

$\mathsf{Solve} : \mathbb{F}_q^{k \times mn} \times \mathbb{F}_q \mapsto (\mathbb{F}_q^{m \times m} \cup \{\perp\}) \times (\mathbb{F}_q^{n \times n} \cup \{\perp\})\}$ takes $\mathbf{G}'_0 \in \mathbb{F}_q^{k \times mn}$ and $a_{m-1,m-1} \in \mathbb{F}_q$ as input and sets $\mathbf{P}_0^{(0)}, \mathbf{P}_1^{(0)}, \ldots \in \mathbb{F}_q^{m \times n} = \mathsf{RowsToMatrices}(\mathbf{G}'_0)$. Consider the following equations with $\mathbf{A}$ and $\mathbf{B}$ unknown:

$$\mathbf{P}_0 \mathbf{B}^{-1} = \mathbf{A} \mathbf{P}_0^{(0)} \tag{1a}$$

$$\mathbf{P}_1 \mathbf{B}^{-1} = \mathbf{A} \mathbf{P}_1^{(0)}, \tag{1b}$$

which gives $2mn$ equations in $m^2 + n^2$ variables over $\mathbb{F}_q$. To derive $\mathbf{A}$ and $\mathbf{B}$, Solve solves the system consisting of of the first $2mn - 1$ equations (for $m = n$, i.e., without the equation for index $[m-1, n-1]$ in (1b)) with $\mathbf{P}_0 = \mathbf{I}_m$ and $\mathbf{P}_1 = \mathbf{U}_m$ for $\mathbf{A} \in \mathbb{F}_q^{m \times m}$ and $\mathbf{B}^{-1} \in \mathbb{F}_q^{n \times n}$ with $\mathbf{A}[m-1; m-1] = a_{m-1,m-1}$, and returns $(\mathbf{A}, \mathbf{B}^{-1})$ if the system has exactly one solution or $(\perp, \perp)$ otherwise.

$\mathsf{H} : \mathcal{B}^* \mapsto \mathcal{B}^{\ell_{\mathsf{digest}}}$ hashes a byte string of arbitrary length to a byte string of length ℓ_{digest}.

$\mathsf{G} : \mathcal{B}^{\ell_{\mathsf{salt}}} \times \mathcal{B}^2 \times \mathcal{B}^{\ell_{\mathsf{tree\_seed}}} \mapsto \mathcal{B}^{\ell_{\mathsf{tree\_seed}}} \times \mathcal{B}^{\ell_{\mathsf{tree\_seed}}}$ takes a salt $\alpha \in \mathcal{B}^{\ell_{\mathsf{salt}}}$, an address a encoded into two bytes, and a seed $\rho \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}}$ as inputs, concatenates and hashes $(\alpha \,|\, a \,|\, \rho)$ to a byte string of length $2\ell_{\mathsf{tree\_seed}}$, and returns the first $\ell_{\mathsf{tree\_seed}}$ and the second $\ell_{\mathsf{tree\_seed}}$ bytes.

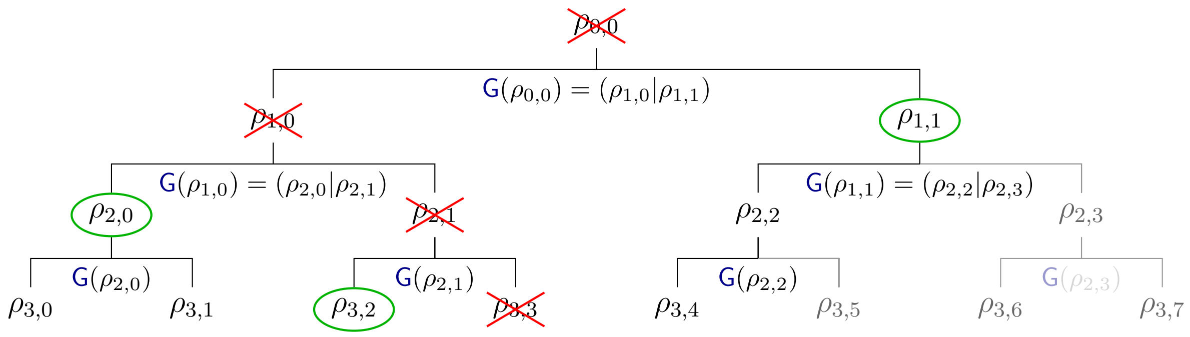

Figure 2: Seed tree example for $t = 5$ and $w = 1$ SeedTree computes the tree structure from root node $\rho_{0,0}$ and generates the seed leafs $\sigma_0 = \rho_{3,0}$ to $\sigma_4 = \rho_{3,4}$. SeedTreeToPath returns the seed-tree path (marked green) for $h_3 > 0$ and $h_i = 0, i \neq 3$. PathToSeedTree recovers the leaf nodes $\rho_{3,0}$, $\rho_{3,1}$, $\rho_{3,2}$, and $\rho_{3,4}$ form the seed-tree path without learning nodes $\rho_{0,0}$, $\rho_{1,0}$, $\rho_{2,1}$, and $\rho_{3,3}$ (marked red).

$\mathsf{SeedTree}_t : \mathcal{B}^{\ell_{\mathsf{tree\_seed}}} \times \mathcal{B}^{\ell_{\mathsf{salt}}} \mapsto (\mathcal{B}^{\ell_{\mathsf{tree\_seed}}})^t$ takes a root seed $\rho_{0,0} \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}}$ and a salt $\alpha \in \mathcal{B}^{\ell_{\mathsf{salt}}}$ as input, computes a binary seed tree of height $\lceil \log_2(t) \rceil$, and returns the first t leaf nodes as seeds $\sigma_0, \dots, \sigma_{t-1} \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}}$. The seed tree is constructed recursively starting form the root with seed $\rho_{0,0}$ on level 0: The seed of length $\mathcal{B}^{\ell_{\mathsf{tree\_seed}}}$ of the ith node on level j is hashed together with the salt α and the node address $a_{i,j}$ as $\mathsf{G}(\alpha, a_{i,j}, \rho_{i,j})$. The two hash outputs $\rho_{2i,j+1}$ and $\rho_{2i+2,j+1}$ of G are assigned to the $2i$th and $(2i+1)$th nodes on level $j+1$. The node address $a_{i,j}$ is computed as $a_{i,j} = 2^j - 1 + i$ and converted to four bytes in little endian order as input to G. Figure 2 shows an example for a seed tree with $t = 5$.

$\mathsf{SeedTreeToPath}_t : \{0, \dots, s-1\}^t \times \mathcal{B}^{\ell_{\mathsf{tree\_seed}}} \times \mathcal{B}^{\ell_{\mathsf{salt}}} \mapsto \mathcal{B}^{\ell_{\mathsf{path}}}$ takes as input an expanded digest $h_0, \dots, h_{t-1} \in \{0, \dots, s-1\}$, a root seed $\rho \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}}$, and a salt $\alpha \in \mathcal{B}^{\ell_{\mathsf{salt}}}$. This function recomputes a seed tree with the same structure as $\mathsf{SeedTree}_t()$. It then for each i with $h_i \neq 0$ removes the seed label of each ith leaf node and deletes the seed labels of all parent nodes up to the tree root node. This function then outputs as seed-tree path sequentially in depth-first order all non-empty labels at the highest level padded with 0-bytes up to length ℓ_{path}.

$\mathsf{PathToSeedTree}_t : \{0, \dots, s-1\}^t \times \mathcal{B}^{\ell_{\mathsf{path}}} \times \mathcal{B}^{\ell_{\mathsf{salt}}} \mapsto \mathcal{B}^{\ell_{\mathsf{tree\_seed}} \times t}$ takes as input an expanded digest $h_0, \dots, h_{t-1} \in \{0, \dots, s-1\}$, a seed-tree path $p \in \mathcal{B}^{\ell_{\mathsf{path}}}$, and a salt $\alpha \in \mathcal{B}^{\ell_{\mathsf{salt}}}$. This function recomputes an empty seed tree (nodes without labels) with the same structure as $\mathsf{SeedTree}_t()$. It then applies the seed-tree path $p \in \mathcal{B}^{\ell_{\mathsf{path}}}$ by labeling the corresponding nodes. It then re-computes the missing higher-level node labels up to the leaf nodes. This function then outputs the labels of the leaf nodes in order (including empty labels) as a sequence of seeds $\sigma_0, \dots, \sigma_{t-1} \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}}$.

3.4 MEDS Operations

The MEDS signature scheme makes use of the supporting operations Compress, Decompress, CompressG, DecompressG, ExpandFqs, ExpandSystMat, π, ExpandInvMat, and $\mathsf{ParseHash}_{s,t,w}$ as defined in Algorithms 1 to 9.

The main operations KeyGen, Sign, and Verify of the MEDS signature scheme are defined in Algorithms 10 to 12.

Algorithm 1: Compress : $\mathbb{F}_q^{*,*} \mapsto \mathcal{B}^*$ — compress matrix

```
Input: matrix $\mathbf{M} \in \mathbb{F}_q^{m' \times n'}$
Output: byte string $b_0, \ldots, b_{\ell_b - 1} \in \mathcal{B}$ with $\ell_b = \lceil m'n' \cdot \mathsf{BitLen}(q)/8 \rceil$
forall $i \in \{0, \ldots, \ell_b - 1\}$ do
    $b_i \in \mathcal{B} \leftarrow 0$
$f_{\text{byte}} \in \mathbb{Z} \leftarrow 0$
$f_{\text{bit}} \in \mathbb{Z} \leftarrow 0$
forall $i \in \{0, \ldots, m' - 1\}$ do
    forall $j \in \{0, \ldots, n' - 1\}$ do
        $c \in \mathbb{Z} \leftarrow 0$
        $v \in \mathbb{Z} \leftarrow \mathsf{Z}(\mathbf{M}[i;j])$
        while $c < \mathsf{BitLen}(q)$ do
            $c' \in \mathbb{Z} \leftarrow \min(8 - f_{\text{bit}}, \mathsf{BitLen}(q) - c)$
            $b_{f_{\text{byte}}} \leftarrow b_{f_{\text{byte}}} + (v \bmod 2^{c'}) \cdot 2^{f_{\text{bit}}}$
            $v \leftarrow \lfloor v / 2^{c'} \rfloor$
            $c \leftarrow c + c'$
            $f_{\text{bit}} \leftarrow f_{\text{bit}} + c'$
            if $f_{bit} = 8$ then
                $f_{\text{bit}} \leftarrow 0$
                $f_{\text{byte}} \leftarrow f_{\text{byte}} + 1$
return $b_0, \ldots, b_{\ell_b - 1} \in \mathcal{B}$
```

Algorithm 2: Decompress : $\mathcal{B}^* \times \mathbb{Z} \times \mathbb{Z} \mapsto \mathbb{F}_q^{*,*}$ — decompress matrix

```
Input: byte string $b_0, \ldots, b_{\ell_b - 1} \in \mathcal{B}$ with $\ell_b = \lceil m'n' \cdot \mathsf{BitLen}(q)/8 \rceil$, $m', n' \in \mathbb{Z}^+$
Output: matrix $\mathbf{M} \in \mathbb{F}_q^{m' \times n'}$
$f_{\text{byte}} \in \mathbb{Z} \leftarrow 0$
$f_{\text{bit}} \in \mathbb{Z} \leftarrow 0$
forall $i \in \{0, \ldots, m' - 1\}$ do
    forall $j \in \{0, \ldots, n' - 1\}$ do
        $c \leftarrow 0$
        $v \leftarrow 0$
        while $c < \mathsf{BitLen}(q)$ do
            $c' \leftarrow \min(8 - f_{\text{bit}}, \mathsf{BitLen}(q) - c)$
            $v \leftarrow v + (\lfloor b_{f_{\text{byte}}} / 2^{f_{\text{bit}}} \rfloor \bmod 2^{c'}) \cdot 2^c$
            $c \leftarrow c + c'$
            $f_{\text{bit}} \leftarrow f_{\text{bit}} + c'$
            if $f_{bit} = 8$ then
                $f_{\text{bit}} \leftarrow 0$
                $f_{\text{byte}} \leftarrow f_{\text{byte}} + 1$
        $\mathbf{M}[i;j] \leftarrow \mathsf{F_q}(v)$
return $\mathbf{M} \in \mathbb{F}_q^{m' \times n'}$
```

Algorithm 3: CompressG : $\mathbb{F}_q^{k\times mn} \mapsto \mathcal{B}^*$ — compress public key matrix

```
Input: matrix $\mathbf{G} \in \mathbb{F}_q^{k\times mn}$
Output: byte string $b_0, \ldots, b_{\ell_b-1} \in \mathcal{B}$ with $\ell_b = \lceil \ell_{\mathbf{G}'} \cdot \mathsf{BitLen}(q)/8 \rceil$ and
        $\ell_{\mathbf{G}'} = (k-2)(mn-k)+n$
1  $\mathbf{G}' \in \mathbb{F}_q^{1\times \ell_{\mathbf{G}'}} \leftarrow \mathbf{0}_{1\times \ell_{\mathbf{G}'}}$
2  $f_{\mathbf{G}'} \leftarrow 0$
3  forall $i \in \{0,\ldots,n-1\}$ do
4      $\mathbf{G}'[0; f_{\mathbf{G}'}] \leftarrow \mathbf{G}[1; mn-n+i]$
5      $f_{\mathbf{G}'} \leftarrow f_{\mathbf{G}'} + 1$
6  forall $i \in \{2,\ldots,k-1\}$ do
7      forall $j \in \{k,\ldots,mn-1\}$ do
8          $\mathbf{G}'[0; f_{\mathbf{G}'}] \leftarrow \mathbf{G}[i; j]$
9          $f_{\mathbf{G}'} \leftarrow f_{\mathbf{G}'} + 1$
10 return $b_0, \ldots, b_{\ell_b-1} \in \mathcal{B} = \mathsf{Compress}(\mathbf{G}')$
```

Algorithm 4: DecompressG : $\mathcal{B}^* \mapsto \mathbb{F}_q^{k\times mn}$ — decompress public key matrix

```
Input: byte string $b \in \mathcal{B}^{\ell_b}$ with
       $\ell_b = \lceil \ell_{\mathbf{G}'} \cdot \mathsf{BitLen}(q)/8 \rceil$ and $\ell_{\mathbf{G}'} = (k-2)(mn-k)+n$
Output: matrix $\mathbf{G} \in \mathbb{F}_q^{k\times mn}$
1  $\mathbf{G}' \in \mathbb{F}_q^{1\times \ell_{\mathbf{G}'}} \leftarrow \mathsf{Decompress}(b, 1, \ell_{\mathbf{G}'})$
2  $\mathbf{G} \in \mathbb{F}_q^{k\times mn} \leftarrow (\mathbf{I}_k | \mathbf{0}_{k\times mn-k})$
3  forall $i \in \{1,\ldots,m-1\}$ do
4      $\mathbf{G}[0; i(n+1)] \leftarrow 1$
5  forall $i \in \{1,\ldots,m-2\}$ do
6      $\mathbf{G}[1; i(n+1)+1] \leftarrow 1$
7  $f_{\mathbf{G}'} \leftarrow 0$
8  forall $i \in \{0,\ldots,n-1\}$ do
9      $\mathbf{G}[1; mn-n+i] \leftarrow \mathbf{G}'[0; f_{\mathbf{G}'}]$
10     $f_{\mathbf{G}'} \leftarrow f_{\mathbf{G}'} + 1$
11 forall $i \in \{2,\ldots,k-1\}$ do
12     forall $j \in \{k,\ldots,mn-1\}$ do
13         $\mathbf{G}[i; j] \leftarrow \mathbf{G}'[0; f_{\mathbf{G}'}]$
14         $f_{\mathbf{G}'} \leftarrow f_{\mathbf{G}'} + 1$
15 return $\mathbf{G} \in \mathbb{F}_q^{k\times mn}$
```

Algorithm 5: ExpandFqs : $\mathcal{B}^*, \mathbb{Z} \mapsto \mathbb{F}_q^*$ — generate field elements from seed

```
Input: seed $\sigma \in \mathcal{B}^{\ell_\sigma}$ of length $\ell_\sigma$,
       integer $\ell \in \mathbb{Z}$
Output: sequence of $\ell$ field elements $a_0, a_1, \ldots, a_{\ell-1} \in \mathbb{F}_q$
1  $b_0, b_1, \dots \in \mathcal{B} \leftarrow \mathsf{XOF}(\sigma, *)$
2  $f \in \mathbb{Z} \leftarrow 0$
3  forall $i \in \{0, \ldots, \ell - 1\}$ do
4      $a_i \in \mathbb{Z} \leftarrow 0$
5      forall $j \in \{0, \ldots, \mathsf{ByteLen}(q) - 1\}$ do
6          $a_i \leftarrow a_i + b_f \cdot 2^{8j}$
7          $f \leftarrow f + 1$
8      $a_i \leftarrow a_i \bmod 2^{\mathsf{BitLen}(q)}$
9      if $a_i \geq q$ then
10         goto line 4
11 return $\mathsf{F_q}(a_0), \mathsf{F_q}(a_1), \ldots, \mathsf{F_q}(a_{\ell-1}) \in \mathbb{F}_q$
```

Algorithm 6: ExpandSystMat : $\mathcal{B}^* \mapsto \mathbb{F}_q^{k \times mn}$ — generate systematic matrix from seed

```
Input: seed $\sigma \in \mathcal{B}^{\ell_\sigma}$ of length $\ell_\sigma$
Output: matrix $\mathbf{M} \in \mathbb{F}_q^{k \times mn}$ in systematic form
1  $a_0, a_1, \ldots, a_{k(mn-k)-1} \in \mathbb{F}_q \leftarrow \mathsf{ExpandFqs}(\sigma, k(mn-k))$
2  $f_a \in \mathbb{Z} \leftarrow 0$
3  $\mathbf{M} \in \mathbb{F}_q^{k \times mn} \leftarrow \mathbf{0}_{k \times mn}$
4  forall $i \in \{0, \ldots, k - 1\}$ do
5      $\mathbf{M}[i; i] \in \mathbb{F}_q \leftarrow 1$
6      forall $j \in \{k, \ldots, mn - 1\}$ do
7          $\mathbf{M}[i; j] \in \mathbb{F}_q \leftarrow a_{f_a}$
8          $f_a \leftarrow f_a + 1$
9  return $\mathbf{M} \in \mathbb{F}_q^{k \times mn}$
```

Algorithm 7: $\pi_{\mathbb{F}_q^{m \times m}, \mathbb{F}_q^{n \times n}} : \mathbb{F}_q^{k \times mn} \mapsto \mathbb{F}_q^{k \times mn}$

```
Input: matrices $\mathbf{A} \in \mathbb{F}_q^{m \times m}, \mathbf{G} \in \mathbb{F}_q^{k \times mn}, \mathbf{B} \in \mathbb{F}_q^{n \times n}$ (notation: $\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G})$)
Output: matrix $\mathbf{G}' \in \mathbb{F}_q^{k \times mn}$
1  $\mathbf{P}_0, \ldots, \mathbf{P}_{k-1} \in \mathbb{F}_q^{m \times n} \leftarrow \mathsf{RowsToMatrices}(\mathbf{G})$
2  for $i \in \{0, \ldots, k - 1\}$ do
3      $\mathbf{P}'_i \in \mathbb{F}_q^{m \times n} \leftarrow \mathbf{A}\mathbf{P}_i\mathbf{B}$
4  $\mathbf{G}' \in \mathbb{F}_q^{k \times mn} \leftarrow \mathsf{MatricesToRows}(\mathbf{P}'_0, \ldots, \mathbf{P}'_{k-1})$
5  return $\mathbf{G}' \in \mathbb{F}_q^{k \times mn}$
```

Algorithm 8: ExpandInvMat : $\mathcal{B}^*, \mathbb{Z} \mapsto \mathbb{F}_q^{*,*}$ — generate invertible matrix from seed

```
Input: seed $\sigma \in \mathcal{B}^{\ell_\sigma}$ of length $\ell_\sigma$, matrix dimension $d \in \mathbb{Z}$
Output: invertible matrix $\mathbf{M} \in \mathrm{GL}_d(q)$
1 $a_0, a_1, \dots \in \mathbb{F}_q \leftarrow \mathsf{ExpandFqs}(\sigma, *)$
2 $f_a \in \mathbb{Z} \leftarrow 0$
3 forall $i \in \{0, \dots, d-1\}$ do
4     forall $j \in \{0, \dots, d-1\}$ do
5         $\mathbf{M}[i; j] \in \mathbb{F}_q \leftarrow a_{f_a}$
6         $f_a \leftarrow f_a + 1$
7 if $\mathbf{M} \notin \mathrm{GL}_d(q)$ then
8     goto line 3
9 return $\mathbf{M} \in \mathrm{GL}_d(q)$
```

Algorithm 9: ParseHash$_{s,t,w}$: $\mathcal{B}^{\ell_{\mathsf{digest}}} \mapsto \{0, \dots, s-1\}^t$ — parse hash to weight-w vector

```
Input: digest $d \in \mathcal{B}^{\ell_{\mathsf{digest}}}$ as byte string,
       parameters $s \in \{2, \dots, 256\},\ t, w \in \mathbb{Z}^+,\ w \ll t$
Output: $h_0, \dots, h_{t-1} \in \{0, \dots, s-1\}$ such that $w$ elements are non-zero
1 $b_0, b_1, \dots \in \mathcal{B}^* \leftarrow \mathsf{XOF}(d, *)$
2 $f_b \leftarrow 0$
3 forall $i \in \{0, \dots, t-1\}$ do
4     $h_i \leftarrow 0$
5 forall $i \in \{0, \dots, w-1\}$ do
6     $f_h \leftarrow 0$
7     forall $j \in \{0, \dots, \mathsf{ByteLen}(t) - 1\}$ do
8         $f_h \leftarrow f_h + (b_{f_b} 2^{8j})$
9         $f_b \leftarrow f_b + 1$
10    $f_h \leftarrow f_h \bmod 2^{\mathsf{BitLen}(t)}$
11    if $f_h \geq t$ or $h_{f_h} > 0$ then
12        goto line 6
13    $h_{f_h} \leftarrow b_{f_b}$
14    $f_b \leftarrow f_b + 1$
15    $h_{f_h} \leftarrow h_{f_h} \bmod 2^{\mathsf{BitLen}(s)}$
16    if $h_{f_h} = 0$ or $h_{f_h} \geq s$ then
17        goto line 13
18 return $h_0, \dots, h_{t-1} \in \{0, \dots, s-1\}$
```

Algorithm 10: MEDS.KeyGen(): key generation

```
Input: —
Output: public key $\mathsf{pk} \in \mathcal{B}^{\ell_{\mathsf{pk}}}$, secret key $\mathsf{sk} \in \mathcal{B}^{\ell_{\mathsf{sk}}}$
1  $\delta \in \mathcal{B}^{\ell_{\mathsf{sec\_seed}}} \leftarrow \mathsf{Randombytes}(\ell_{\mathsf{sec\_seed}})$
2  $\sigma_{\mathbf{G}_0} \in \mathcal{B}^{\ell_{\mathsf{pub\_seed}}}, \sigma \in \mathcal{B}^{\ell_{\mathsf{sec\_seed}}} \leftarrow \mathsf{XOF}(\delta, \ell_{\mathsf{pub\_seed}}, \ell_{\mathsf{sec\_seed}})$
3  $\mathbf{G}_0 \in \mathbb{F}_q^{k \times mn} \leftarrow \mathsf{ExpandSystMat}(\sigma_{\mathbf{G}_0})$
4  forall $i \in \{1, \dots, s-1\}$ do
5      $\sigma_a, \sigma_{\mathbf{T}_i}, \sigma \in \mathcal{B}^{\ell_{\mathsf{sec\_seed}}} \leftarrow \mathsf{XOF}(\sigma, \ell_{\mathsf{sec\_seed}}, \ell_{\mathsf{sec\_seed}}, \ell_{\mathsf{sec\_seed}})$
6      $\mathbf{T}_i, \in \mathrm{GL}_k(q) \leftarrow \mathsf{ExpandInvMat}(\sigma_{\mathbf{T}_i}, k)$
7      $a_{m-1,m-1} \in \mathbb{F}_q \leftarrow \mathsf{ExpandFqs}(\sigma_a, 1)$
8      $\mathbf{G}_0' \in \mathbb{F}_q^{k \times mn} \leftarrow \mathbf{T}_i \mathbf{G}_0$
9      $\check{\mathbf{A}}_i \in \mathbb{F}_q^{m \times m} \cup \{\perp\}, \check{\mathbf{B}}_i \in \mathbb{F}_q^{n \times n} \cup \{\perp\} \leftarrow \mathsf{Solve}(\mathbf{G}_0', a_{m-1,m-1})$
10     if $(\check{\mathbf{A}}_i = \perp$ and $\check{\mathbf{B}}_i = \perp)$ or $\check{\mathbf{A}}_i \notin \mathrm{GL}_m(q)$ or $\check{\mathbf{B}}_i \notin \mathrm{GL}_n(q)$ then
11         goto line 5
12     $\mathbf{A}_i, \mathbf{A}_i^{-1} \in \mathrm{GL}_m(q) \leftarrow \check{\mathbf{A}}_i, \check{\mathbf{A}}_i^{-1}$
13     $\mathbf{B}_i, \mathbf{B}_i^{-1} \in \mathrm{GL}_n(q) \leftarrow \check{\mathbf{B}}_i^{-1}, \check{\mathbf{B}}_i$
14     $\mathbf{G}_i \in \mathbb{F}_q^{k \times mn} \leftarrow \pi_{\mathbf{A}_i, \mathbf{B}_i}(\mathbf{G}_0)$
15     $\mathbf{G}_i \in \mathbb{F}_q^{k \times mn} \cup \{\perp\} \leftarrow \mathsf{SF}(\mathbf{G}_i)$
16     if $\mathbf{G}_i = \perp$ then
17         goto line 5
18 $\mathsf{pk} \in \mathcal{B}^{\ell_{\mathsf{pk}}} \leftarrow (\sigma_{\mathbf{G}_0} \,|\, \mathsf{CompressG}(\mathbf{G}_1) \,|\, \dots \,|\, \mathsf{CompressG}(\mathbf{G}_{s-1}))$
19 $\mathsf{sk} \in \mathcal{B}^{\ell_{\mathsf{sk}}} \leftarrow$
     $(\delta \,|\, \sigma_{\mathbf{G}_0} \,|\, \mathsf{Compress}(\mathbf{A}_1^{-1}) \,|\, \dots \,|\, \mathsf{Compress}(\mathbf{A}_{s-1}^{-1}) \,|\, \mathsf{Compress}(\mathbf{B}_1^{-1}) \,|\, \dots \,|\, \mathsf{Compress}(\mathbf{B}_{s-1}^{-1}))$
20 return $\mathsf{pk}, \mathsf{sk}$
```

Algorithm 11: MEDS.Sign(): signing

```
Input: secret key $\mathsf{sk} \in \mathcal{B}^{\ell_{\mathsf{sk}}}$, message $m \in \mathcal{B}^{\ell_m}$
Output: signed message $m_s \in \mathcal{B}^{\ell_{\mathsf{sig}}+\ell_m}$
1  $f_{\mathsf{sk}} \leftarrow \ell_{\mathsf{sec\_seed}}$
2  $\sigma_{\mathbf{G}_0} \leftarrow \mathsf{pk}[f_{\mathsf{sk}}, f_{\mathsf{sk}} + \ell_{\mathsf{pub\_seed}} - 1]$
3  $f_{\mathsf{sk}} \leftarrow f_{\mathsf{sk}} + \ell_{\mathsf{pub\_seed}}$
4  $\mathbf{G}_0 \in \mathbb{F}_q^{k \times mn} \leftarrow \mathsf{ExpandSystMat}(\sigma_{\mathbf{G}_0})$
5  forall $i \in \{1, \dots, s-1\}$ do
6      $\mathbf{A}_i^{-1} \in \mathbb{F}_q^{m \times m} \leftarrow \mathsf{Decompress}(\mathsf{sk}[f_{\mathsf{sk}}, f_{\mathsf{sk}} + \ell_{\mathbb{F}_q^{m \times m}}], m, m)$
7      $f_{\mathsf{sk}} \leftarrow f_{\mathsf{sk}} + \ell_{\mathbb{F}_q^{m \times m}}$
8  forall $i \in \{1, \dots, s-1\}$ do
9      $\mathbf{B}_i^{-1} \in \mathbb{F}_q^{n \times n} \leftarrow \mathsf{Decompress}(\mathsf{sk}[f_{\mathsf{sk}}, f_{\mathsf{sk}} + \ell_{\mathbb{F}_q^{n \times n}}], n, n)$
10     $f_{\mathsf{sk}} \leftarrow f_{\mathsf{sk}} + \ell_{\mathbb{F}_q^{n \times n}}$
11 $\delta \in \mathcal{B}^{\ell_{\mathsf{sec\_seed}}} \leftarrow \mathsf{Randombytes}(\ell_{\mathsf{sec\_seed}})$
12 $\rho \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}}, \alpha \in \mathcal{B}^{\ell_{\mathsf{salt}}} \leftarrow \mathsf{XOF}(\delta, \ell_{\mathsf{tree\_seed}}, \ell_{\mathsf{salt}})$
13 $\sigma_0, \dots, \sigma_{t-1} \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}} \leftarrow \mathsf{SeedTree}_t(\rho, \alpha)$
14 forall $i \in \{0, \dots, t-1\}$ do
15     $\sigma_{\tilde{\mathbf{A}}_i}, \sigma_{\tilde{\mathbf{B}}_i}, \sigma_i \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}} \leftarrow \mathsf{XOF}(\sigma_i, \ell_{\mathsf{tree\_seed}}, \ell_{\mathsf{tree\_seed}}, \ell_{\mathsf{tree\_seed}})$
16     $\tilde{\mathbf{A}}_i \in \mathrm{GL}_m(q) \leftarrow \mathsf{ExpandInvMat}(\sigma_{\tilde{\mathbf{A}}_i}, m)$
17     $\tilde{\mathbf{B}}_i \in \mathrm{GL}_n(q) \leftarrow \mathsf{ExpandInvMat}(\sigma_{\tilde{\mathbf{B}}_i}, n)$
18     $\tilde{\mathbf{G}}_i \in \mathbb{F}_q^{k \times mn} \leftarrow \pi_{\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i}(\mathbf{G}_0)$
19     $\tilde{\mathbf{G}}_i \in \mathbb{F}_q^{k \times mn} \cup \{\perp\} \leftarrow \mathsf{SF}(\tilde{\mathbf{G}}_i)$
20     if $\tilde{\mathbf{G}}_i = \perp$ then
21         goto line 15
22 $d \in \mathcal{B}^{\ell_{\mathsf{digest}}} \leftarrow \mathsf{H}(\mathsf{Compress}(\tilde{\mathbf{G}}_0[; k, mn-1]) \mid \dots \mid \mathsf{Compress}(\tilde{\mathbf{G}}_{t-1}[; k, mn-1]) \mid m)$
23 $h_0, \dots, h_{t-1} \in \{0, \dots, s-1\} \leftarrow \mathsf{ParseHash}_{s,t,w}(d)$
24 $f_v \leftarrow 0$
25 forall $i \in \{0, \dots, t-1\}$ do
26     if $h_i > 0$ then
27         $\mu_i \in \mathbb{F}_q^{m \times m} \leftarrow \tilde{\mathbf{A}}_i \cdot \mathbf{A}_{h_i}^{-1}$
28         $\nu_i \in \mathbb{F}_q^{n \times n} \leftarrow \mathbf{B}_{h_i}^{-1} \cdot \tilde{\mathbf{B}}_i$
29         $v_{f_v} \in \mathcal{B}^{\ell_{\mathbb{F}_q^{m \times m}} + \ell_{\mathbb{F}_q^{n \times n}}} \leftarrow (\mathsf{Compress}(\mu_i) \mid \mathsf{Compress}(\nu_i))$
30         $f_v \leftarrow f_v + 1$
31 $p \in \mathcal{B}^{\ell_{\mathsf{path}}} \leftarrow \mathsf{SeedTreeToPath}_t(h_0, \dots, h_{t-1}, \rho, \alpha)$
32 return $m_s \in \mathcal{B}^{w(\ell_{\mathbb{F}_q^{m \times m}} + \ell_{\mathbb{F}_q^{n \times n}}) + \ell_{\mathsf{path}} + \ell_{\mathsf{digest}} + \ell_{\mathsf{salt}} + \ell_m = \ell_{\mathsf{sig}} + \ell_m} = (v_0 \mid \dots \mid v_{w-1} \mid p \mid d \mid \alpha \mid m)$
```

Algorithm 12: MEDS.Verify(): verification

```
Input: public key $\mathsf{pk} \in \mathcal{B}^{\ell_{\mathsf{pk}}}$, signed message $m_s \in \mathcal{B}^{\ell_{\mathsf{sig}}+\ell_m}$
Output: message $m \in \mathcal{B}^{\ell}_m$ or $\perp$
1  $\sigma_{\mathbf{G}_0} \leftarrow \mathsf{pk}[0, \ell_{\mathsf{pub\_seed}} - 1]$
2  $\mathbf{G}_0 \in \mathbb{F}_q^{k\times mn} \leftarrow \mathsf{ExpandSystMat}(\sigma_{\mathbf{G}_0})$
3  $f_{\mathsf{pk}} \leftarrow \ell_{\mathsf{pub\_seed}}$
4  forall $i \in \{1,\dots,s-1\}$ do
5      $\mathbf{G}_i \in \mathbb{F}_q^{k\times mn} \leftarrow \mathsf{DecompressG}(\mathsf{pk}[f_{\mathsf{pk}}, f_{\mathsf{pk}} + \ell_{G_i}])$
6      $f_{\mathsf{pk}} \leftarrow f_{\mathsf{pk}} + \ell_{G_i}$
7  $p \in \mathcal{B}^{\ell_{\mathsf{path}}} \leftarrow m_s[\ell_{\mathsf{sig}} - \ell_{\mathsf{digest}} - \ell_{\mathsf{salt}} - \ell_{\mathsf{path}}, \ell_{\mathsf{sig}} - \ell_{\mathsf{digest}} - \ell_{\mathsf{salt}} - 1]$
8  $d \in \mathcal{B}^{\ell_{\mathsf{digest}}}, \alpha \in \mathcal{B}^{\ell_{\mathsf{salt}}}, m \in \mathcal{B}^{*} \leftarrow$
     $m_s[\ell_{\mathsf{sig}} - \ell_{\mathsf{digest}} - \ell_{\mathsf{salt}}, \ell_{\mathsf{sig}} - \ell_{\mathsf{salt}} - 1], m_s[\ell_{\mathsf{sig}} - \ell_{\mathsf{salt}}, \ell_{\mathsf{sig}} - 1], m_s[\ell_{\mathsf{sig}},]$
9  $h_0,\dots,h_{t-1} \in \{0,\dots,s-1\} \leftarrow \mathsf{ParseHash}_{s,t,w}(d)$
10 $\sigma_0,\dots,\sigma_{t-1} \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}} \leftarrow \mathsf{PathToSeedTree}_t(h_0,\dots,h_{t-1},p,\alpha)$
11 $f_{m_s} \leftarrow 0$
12 forall $i \in \{0,\dots,t-1\}$ do
13     if $h_i > 0$ then
14         $\mu_i \in \mathbb{F}_q^{m\times m} \leftarrow \mathsf{Decompress}(m_s[f_{m_s}, f_{m_s} + \ell_{\mathbb{F}_q^{m\times m}} - 1], m, m)$
15         $\nu_i \in \mathbb{F}_q^{n\times n} \leftarrow \mathsf{Decompress}(m_s[f_{m_s} + \ell_{\mathbb{F}_q^{m\times m}}, f_{m_s} + \ell_{\mathbb{F}_q^{m\times m}} + \ell_{\mathbb{F}_q^{n\times n}} - 1], n, n)$
16         $f_{m_s} \leftarrow f_{m_s} + \ell_{\mathbb{F}_q^{m\times m}} + \ell_{\mathbb{F}_q^{n\times n}};$
17         $\hat{\mathbf{G}}_i \in \mathbb{F}_q^{k\times mn} \leftarrow \pi_{\mu_i,\nu_i}(\mathbf{G}_{h_i})$
18         $\hat{\mathbf{G}}_i \in \mathbb{F}_q^{k\times mn} \cup \{\perp\} \leftarrow \mathsf{SF}(\hat{\mathbf{G}}_i)$
19         if $\hat{\mathbf{G}}_i = \perp$ then
20             return $\perp$
21     else
22         $\sigma_{\hat{\mathbf{A}}_i}, \sigma_{\hat{\mathbf{B}}_i}, \sigma_i \in \mathcal{B}^{\ell_{\mathsf{tree\_seed}}} \leftarrow \mathsf{XOF}(\sigma_i, \ell_{\mathsf{tree\_seed}}, \ell_{\mathsf{tree\_seed}}, \ell_{\mathsf{tree\_seed}})$
23         $\hat{\mathbf{A}}_i \in \mathrm{GL}_m(q) \leftarrow \mathsf{ExpandInvMat}(\sigma_{\hat{\mathbf{A}}_i}, m)$
24         $\hat{\mathbf{B}}_i \in \mathrm{GL}_n(q) \leftarrow \mathsf{ExpandInvMat}(\sigma_{\hat{\mathbf{B}}_i}, n)$
25         $\hat{\mathbf{G}}_i \in \mathbb{F}_q^{k\times mn} \leftarrow \pi_{\hat{\mathbf{A}}_i,\hat{\mathbf{B}}_i}(\mathbf{G}_0)$
26         $\hat{\mathbf{G}}_i \in \mathbb{F}_q^{k\times mn} \cup \{\perp\} \leftarrow \mathsf{SF}(\hat{\mathbf{G}}_i)$
27         if $\hat{\mathbf{G}}_i = \perp$ then
28             goto line 22
29 $d' \in \mathcal{B}^{\ell_{\mathsf{digest}}} \leftarrow \mathsf{H}(\mathsf{Compress}(\hat{\mathbf{G}}_0[;k,mn-1]) \mid \dots \mid \mathsf{Compress}(\hat{\mathbf{G}}_{t-1}[;k,mn-1]) \mid m)$
30 if $d = d'$ then
31     return $m$
32 else
33     return $\perp$
```

4 Security and Known Attacks (2.B.4/2.B.5)

4.1 Hardness Assumption

The security of MEDS is based on the MCE problem, which has been studied in several works [CDG20, RST22, CNP$^+$23]. It is now known to be at least as hard as the Code Equivalence problem in the Hamming metric [CDG20] and equivalent to the homogeneous version of the Quadratic Maps Linear Equivalence problem (QMLE) [RST22] and the alternating trilinear form equivalence problem [GQT22, TDJ$^+$22]. As shown in [GQ23] these three problems are all Tensor Isomorphism (TI)-complete. The entire TI class is believed to be hard, even for quantum adversaries. See a summary of the known algorithms in Section 4.3.

4.2 Provable Security

If we apply only the Fiat-Shamir transform on the MEDS Sigma protocol from Figure 1, we obtain a "non-optimized" version of MEDS. The security of the non-optimized version of MEDS was proven in [CNP$^+$23] in the random oracle model. A detailed statement of the security claim is given in the following theorem:

Theorem 4.1. *The non-optimized version of MEDS is EUF-CMA-secure in the random oracle model, if the following conditions are satisfied:*

- *the search version of the* MCE *problem is intractable in the average case,*
- *the hash function* H, *is modeled as a random oracle,*
- *the hash function* G, *is modeled as a random oracle,*
- *the function* XOF *is modeled as a random oracle.*

The proof follows the standard approach for showing the security of Fiat-Shamir signatures, and is therefore omitted.

Since recently, there exist techniques for showing the security of Fiat-Shamir signatures in the QROM [DFMS19, LZ19], under some mild assumptions. For example, [DFMS19] requires the property of computationally unique responses as defined in [KLS18]. MEDS satisfies this property under mild plausible assumptions. As a matter of fact, under the same assumption, MEDS satisfies the perfect unique response property as defined by Unruh in [Unr12]. For this we need to look at the automorphism group of a given matrix code.

An *automorphism* of a matrix code $\mathcal{C}$ is a map $(\mathbf{A}, \mathbf{B}) : \mathbb{F}_q^{m\times n} \to \mathbb{F}_q^{m\times n}$ so that for each $\mathbf{C} \in \mathcal{C}$, we get $\mathbf{ACB} \in \mathcal{C}$, where $A \in \mathrm{GL}_m(q)$ and $B \in \mathrm{GL}_n(q)$. The *automorphism group* of $\mathcal{C}$ contains all the automorphisms of $\mathcal{C}$. If the automorphism group contains only the maps $(\lambda\mathbf{I}, \nu\mathbf{I})$ for constants $\lambda, \nu \in \mathbb{F}_q^*$, we say the automorphism group is trivial.

If we consider the map $(\mathbf{A}, \mathbf{B}) : \mathcal{C} \to \mathcal{C}$ projectively, a trivial automorphism group contains only the map $(\mathbf{I}, \mathbf{I})$. This is typically the setting that we have even when looking at the MCE problem, where one solution $(\mathbf{A}, \mathbf{B})$, naturally yields as solutions all $(\lambda\mathbf{A}, \nu\mathbf{B})$ for constants $\lambda, \nu \in \mathbb{F}_q^*$. Projectively, however, this is considered as a single solution.

As discussed in [RST22], experimental evidence, as well as results for other algebraic structures, suggests that the automorphism group of a randomly chosen matrix code is trivial, and if assumed trivial, several reductions between hard problems are very tight. Therefore, it is natural to also make this very plausible assumption here.

Assumption Aut: The automorphism group of a randomly chosen matrix code is trivial with overwhelming probability for sufficiently large code length $m \times n$.

We have thus the following based on the results from [DFMS19]:

Theorem 4.2. *The non-optimized version of MEDS is EUF-CMA-secure in the quantum random oracle model, if in addition to the conditions from Theorem 4.1, assumption* Aut *holds.*

We omit a detailed proof, but we note that all of the required properties from [DFMS19] are satisfied by the construction of MEDS. It is only the property of a trivial automorphism group that is required to show that the scheme satisfies the perfect unique response property.

The previous theorems describe the security of the non-optimized version of MEDS. We do however apply several optimizations in order to improve performance and reduce signature size. We discuss separately the implication of these on the security of MEDS.

- **Multiple Public Keys + Fixed-Weight Challenge Strings**. The two optimizations are a common strategy for reducing signature sizes, and have been used in several other signature schemes [Beu19, Ran20, BBPS21]. For example, LESS [BBPS21] applies exactly the same optimizations and it was shown that they do not change the security claims of the scheme. Since LESS is also based on code-equivalence, but in the Hamming metric, the proofs given in [BBPS21] apply also to MEDS, with appropriate natural changes. We refer the reader to [BBPS21] for details.

- **Seed tree**. A comprehensive description for seed trees can be found in [BKP20a] and a ROM proof in [BKP20b, Sec. 2.7]. The adaption of the proof to the QROM setting is to the best of our knowledge still open work. The proof for the seed tree in [BKP20b] requires a salt and domain separation using node addresses to ward of multi-target collision attacks. Both of these are implemented in MEDS.

- **Public Key Compression**. When applying this technique, since all the matrices are chosen uniformly at random, the procedure is equivalent to first choosing two secret codewords $\mathbf{P}_0^{(0)}, \mathbf{P}_1^{(0)}$ from $\mathcal{C}_0$ and a secret map $(\mathbf{A}_i, \mathbf{B}_i)$ and finding the public codewords $\mathbf{P}_0^{(i)}, \mathbf{P}_1^{(i)}$ from $\mathcal{C}_i$, which in turn is equivalent to the procedure in the non-optimized version of MEDS.

  To reduce the computational effort, we further take $\mathbf{P}_0^{(i)}$ and $\mathbf{P}_1^{(i)}$ to be of specific form (the identity matrix, and the identity shifted by one position to the right, respectively), and the same for all $i \in \{0, 1, \ldots, s-1\}$. The question is whether having a specific form of these public matrices makes the MCE problem easier to solve. This is however not the case, as we see from the following argument.

  Suppose there exists an adversary $\mathcal{A}$ with non-negligible advantage against MCE with specified $\mathbf{P}_0^{(i)}$ and $\mathbf{P}_1^{(i)}$. Then, an adversary $\mathcal{B}$ given an MCE instance $\mathsf{MCE}(\mathcal{C}_0, \mathcal{C}_1)$ can first use the above procedure to construct $\mathcal{C}_2 = \phi_1(\mathcal{C}_0)$ where $\mathcal{C}_2$ has specified codewords $\mathbf{P}_0^{(i)}$ and $\mathbf{P}_1^{(i)}$. Now, $\mathcal{B}$ gives the instance $\mathsf{MCE}(\mathcal{C}_1, \mathcal{C}_2)$ to $\mathcal{A}$ who outputs the isometry ϕ_2 between the two codes with non-negligible advantage. $\mathcal{B}$ can now use the isometries ϕ_1 and ϕ_2 to easily find the isometry $\phi = \phi_1 \circ \phi_2^{-1}$ between $\mathcal{C}_0$ and $\mathcal{C}_1$ with essentially the same advantage as $\mathcal{A}$.

4.3 Known Attacks

The known attacks against MEDS are basically attacks against the underlying hard problem MCE since the construction of the signature scheme from MCE is provably secure. There are three main types of attacks against MCE:

1. Algebraic attacks
2. Leon-like algorithms adapted to the rank metric
3. Graph-theoretic algorithms for solving polynomial equivalence problems

The state-of-the-art of these attacks is developed and detailed in [CDG20, RST22, CNP+23]. Some similar ideas were developed in the cryptanalysis of ATFE [TDJ+22, Beu22]. We provide here a summary of the known cryptanalytical results, but we also further improve the algebraic approach and the Leon-like algorithm from [CNP+23]. At the end, we discuss the applicability of the ideas of Beullens [Beu22] in the context of MCE.

4.3.1 Algebraic Attacks

The algebraic attack presented here is an extension of the improved modeling from [CNP+23, §6.2]. Consider the trilinear map $\mathcal{C} : \mathbb{F}_q^n \times \mathbb{F}_q^m \times \mathbb{F}_q^k \to \mathbb{F}_q$ corresponding to a matrix code in terms of its basis given by $\mathcal{C}(x, y, z) = \sum_{i,j,k} \mathbf{C}_{ij}^{(k)} x_i y_j z_k$. The MCE problem can then be rephrased as the 3-TI problem of finding $\mathbf{A} \in \mathrm{GL}_n(q)$, $\mathbf{B} \in \mathrm{GL}_m(q)$ and $\mathbf{T} \in \mathrm{GL}_k(q)$ such that

$$\mathcal{C}(\mathbf{Ax},\ \mathbf{By},\ \mathbf{Tz}) = \mathcal{D}(\mathbf{x},\ \mathbf{y},\ \mathbf{z})\ .$$

We will consider the following system of equations;

$$\mathcal{C}(\mathbf{Ax},\ \mathbf{By},\ \mathbf{z}) = \mathcal{D}(\mathbf{x},\ \mathbf{y},\ \mathbf{T}^{-1}\mathbf{z}) \tag{2a}$$
$$\mathcal{C}(\mathbf{Ax},\ \mathbf{y},\ \mathbf{Tz}) = \mathcal{D}(\mathbf{x},\ \mathbf{B}^{-1}\mathbf{y},\ \mathbf{z}) \tag{2b}$$
$$\mathcal{C}(\mathbf{x},\ \mathbf{By},\ \mathbf{Tz}) = \mathcal{D}(\mathbf{A}^{-1}\mathbf{x},\ \mathbf{y},\ \mathbf{z})\ . \tag{2c}$$

The modeling [CNP+23, §6.2] allows us to remove the inverse matrix $\mathbf{T}^{-1}$ from the equations (2a). This results in a bilinear system of $k(nm - k)$ equations in the n^2 variables a_{ij} and m^2 variables b_{ij}. In this modeling we do the same but then for all equations (2a), (2b), and (2c).

Let us revisit how to eliminate the inverse matrices. Consider matrices $\omega \in \mathbb{F}_q^{n\times m}$ with decomposition $\omega = \sum_\alpha \mathbf{v}_\alpha^T \mathbf{w}_\alpha$ such that $\sum_\alpha \mathcal{D}(\mathbf{v}_\alpha, \mathbf{w}_\alpha, \mathbf{z}) = 0$ for all $\mathbf{z} \in \mathbb{F}_q^k$. Note, by trilinearity of $\mathcal{D}$, that this property is independent of decomposition[2]. By the rank-nullity theorem these matrices span a subspace of dimension at least $nm - k$ and a basis $\{\omega_\beta\}_\beta$ is easily computed. Evaluate equation (2a) according to such $\omega_\beta = \sum_\alpha \mathbf{v}_{\beta,\alpha}^T \mathbf{w}_{\beta,\alpha}$ and we get the following $k(nm-k)$ linear independent equations;

$$\sum_\alpha \mathcal{C}(\mathbf{Av}_{\beta,\alpha},\ \mathbf{Bw}_{\beta,\alpha},\ \mathbf{z}) = 0, \quad \forall\ \beta, \mathbf{z}\ . \tag{3}$$

In fact, there exists a basis $\{\omega_\beta\}_\beta$ such that each ω_β has only $k + 1$ non-zero entries. When using this basis each of the above equations has at most $nm(k + 1)$ terms. The same trick can be applied to (2b) and (2c) to obtain a tri-homogeneous system in the variable sets $(\{a_{ij}\}_{i,j}, \{b_{ij}\}_{i,j}, \{t_{ij}\}_{i,j})$. The resulting system has $k(nm - k)$, $m(nk - m)$ and $n(mk - n)$ equations in tri-degrees $(1, 1, 0)$, $(1, 0, 1)$ and $(0, 1, 1)$.

[2]In fact we can uniquely extend $\mathcal{D}$ to $\mathbb{F}_q^{n\times m} \times \mathbb{F}_q^k \to \mathbb{F}_q$ and consider $\mathcal{D}(\omega, \mathbf{z})$, in which case independence of composition is clear.

This system is not generic however, there are syzygies in tri-degree $(1,1,1)$. Take linear combinations of equations (3) multiplied by t_{ij} as follows:

$$\sum_j t_{ji} \sum_\alpha \mathcal{C}(\mathbf{A}\mathbf{v}_{\beta,\alpha},\ \mathbf{B}\mathbf{w}_{\beta,\alpha},\ \mathbf{e}_j) = \sum_\alpha \mathcal{C}(\mathbf{A}\mathbf{v}_{\beta,\alpha},\ \mathbf{B}\mathbf{w}_{\beta,\alpha},\ \mathbf{T}\mathbf{e}_i) = 0, \quad \forall\, i, \beta$$

This can be done for the other tri-degree types as well and all these equations are $\mathcal{C} \circ (\mathbf{A}, \mathbf{B}, \mathbf{T})$ evaluated on a 3-way array. However, there are only nmk three-way arrays of which $nmk - 1$ actually lie in the kernel of $\mathcal{D}$. A rank-nullity argument then shows that there are at least $n(mk-n) + m(nk-m) + k(nm-k) - nmk + 1$ syzygies in tri-degree $(1,1,1)$. We conjecture that the corresponding tri-Hilbert series of this system is given, up to tri-degree (n,m,k), by

$$\mathcal{H}(r,s,t) = \frac{(1-rs)^{n(mk-n)} \cdot (1-st)^{m(nk-m)} \cdot (1-tr)^{k(nm-k)} \cdot (1-rts)^{-(2nmk-n^2-m^2-k^2+1)}}{(1-r)^{n^2} \cdot (1-s)^{m^2} \cdot (1-t)^{k^2}}.$$

Here, the coefficient of $r^\alpha s^\beta t^\gamma$ equals the right nullity of the Macaulay matrix of tri-degree (α, β, γ) obtained from the system. Furthermore, the generating function for the number of monomials in a certain tri-degree is given by:

$$\mathcal{M}(r,s,t) = \frac{1}{(1-r)^{n^2}(1-s)^{m^2}(1-t)^{k^2}}$$

Now using an algorithm such as block Wiedemann we get a complexity of

$$\min_{\substack{(\alpha,\beta,\gamma) \prec (n,m,k) \\ [r^\alpha s^\beta t^\gamma]\mathcal{H} \leq 1}} 3 \cdot [r^\alpha s^\beta t^\gamma]\mathcal{M}^2 \cdot nm(k+1)\ .$$

To hybridize this approach, e.g. to parallelize or to Groverize, one can use the structure of the equations as done previously in [CNP+23, §6.2]. Assuming that $n = m = k$ one can fix 2 rows of $\mathbf{A}$, so $2n$ variables (a_{ij}), to obtain a linear system for $\mathbf{B}$. This results in a complexity of $\mathcal{O}(q^{2n} n^6)$. Note that even though we have a larger system than before, the extra equations (2b) and (2c) play no role here. If one manages to Groverize this, a complexity of $\mathcal{O}(q^n n^6)$ can be obtained. Even though this is asymptotically better than the complexity of the full algebraic attack, this performs worse for the proposed parameter sets due to the field size. Furthermore, due to the sheer amount of variables, fixing fewer than $2n$ variables will not substantially alter the used tri-degree and will hence not lead to improvements that are significant enough to balance out the enumeration cost. We conclude that in this algebraic attack, a quantum version will not outperform the classical attack.

4.3.2 Leon-like Algorithm

Leon's algorithm [Leo82] is a well-known algorithm for solving code equivalence in the Hamming metric, which works by finding low-weight codewords in both codes and "matching these up" in the right way to recover the isometry. Analogues in the rank metric [CNP+23, §6.3] similarly require finding low-rank codewords in both $\mathcal{C}$ and $\mathcal{D}$, and finding collisions between these sets of low-rank codewords. In [CNP+23], the authors propose to use a two-collision method and they argue that in this case, both the birthday-based approach and the deterministic one have the same complexity. Here, we extend the analysis of the two-collision approach and then we propose an algorithm that requires only one collision and allows for an improved birthday attack.

The Leon-like algorithm is comprised of two steps. First, we build two lists of codewords of rank r in $\mathcal{C}$ and $\mathcal{D}$ respectively. Looking for low-rank codewords can be modeled as an instance of the MinRank [Cou01, FLP08] problem for k matrices of size $m \times n$. We will denote by $\mathsf{MR}_{(n,m,k,r)}$ the cost of solving this MinRank instance using state-of-the-art algorithms such as the Kipnis-Shamir modeling [KS99] or Bardet *et al.*'s modeling [BBC+20]. The expected number of codewords of rank r when $n = m = k$ (which is valid for all our parameter sets) can be approximated by $C(r) \approx q^{r(2n-r)-n^2+n}$, so the instance of MinRank that we are solving is expected to have a solution space of dimension $\sigma = r(2n-r) - n^2 + n$. Thus, in the algebraic modeling of MinRank, we can assign up to σ variables and still expect to have a solution with high probability. This results in an easier MinRank instance of cost $\mathsf{MR}_{(n,n,n-\sigma,r)}$. In general, as an estimate of the complexity of this step, we have $\mathsf{MR}_{(n,r)} = \min\{\mathsf{MR}_{(n,n,n,r)}, q^\sigma \cdot \mathsf{MR}_{(n,n,n-\sigma,r)}\}$.

The second step of the algorithm is the collision search in the lists. A collision $(\mathbf{C}_1, \mathbf{D}_1)$ produces linear relations of the form $\mathbf{A}^{-1}\mathbf{D}_1 = \mathbf{C}_1\mathbf{B}$ in the unknown coefficients of $\mathbf{A}^{-1}$ and $\mathbf{B}$, so having two collisions yields a system of $2n^2$ equations in $2n^2$ variables. However, since $\mathbf{C}_1$, $\mathbf{C}_2$, $\mathbf{D}_1$ and $\mathbf{D}_2$ all have rank r, the number of linearly independent equations is at most $2n^2 - 2(n-r)^2$. In the case where $n = m = k$, the target rank r can be approximated by $r = n - \sqrt{n}$. Then, the number of expected linearly independent equations is approximately $2n^2 - 2n$. Using only these linear equations is not enough to solve the system efficiently, as we would have to enumerate $2n$ variables. However, the hybrid approach hinted in [CNP+23] does have a polynomial time complexity. Concretely, we can use the following linear equations

$$\begin{cases} \mathbf{A}\mathbf{C}_1 = \mathbf{D}_1\mathbf{B}^{-1} \\ \mathbf{A}\mathbf{C}_2 = \mathbf{D}_2\mathbf{B}^{-1} \\ \mathbf{A}^{-1}\mathbf{D}_1 = \mathbf{C}_1\mathbf{B} \\ \mathbf{A}^{-1}\mathbf{D}_2 = \mathbf{C}_2\mathbf{B} \end{cases} \tag{4}$$

From the first two we derive a solution space for $\mathbf{A}$ of dimension $2n$, and from the last two equations we obtain the same for $\mathbf{B}$. Then, we use the algebraic minors-like modeling from [CNP+23] to restrict the solution space further. Recall that in this modeling, we have $n(n^2-n)$ equations that are bilinear in $\mathbf{A}$ and $\mathbf{B}$. Substituting these by the bases of the solution spaces of $\mathbf{A}$ and $\mathbf{B}$ to the linear equations in (4), we obtain a system of $n(n^2-n)$ bilinear equations in $4n$ variables. The system is homogenous and $(2n, 2n)$ bilinear, so it contains $4n^2$ degree-two monomials. It can thus be linearized and solved efficiently through Gaussian elimination in $(4n^2)^\omega$ operations. The overall complexity of the two-collision deterministic approach follows from the analysis in [CNP+23] and, for a target rank r, it amounts to

$$\mathcal{O}(\max(\mathsf{MR}_{(n,r)}, q^{2\sigma}n^{2\omega})).$$

In the one-collision approach, following a birthday argument, we build lists of size $\sqrt{C(r)}$ elements. Recall that for a collision $(\mathbf{C}_1, \mathbf{D}_1)$ we have that $\mathbf{A}\mathbf{C}_1\mathbf{B} = \mathbf{D}_1$. Let $\mathbf{v}$ be a vector in the kernel of $\mathbf{D}_1$. Then $\mathbf{A}\mathbf{C}_1\mathbf{B}\mathbf{v} = 0$, so $\mathbf{B}\mathbf{v}$ is in the kernel of $\mathbf{C}_1$. Since the kernels of $\mathbf{C}_1$ and $\mathbf{D}_1$ are of dimension $(n-r)$, this argument yields $r(n-r)$ linear equations in the $\mathbf{B}$ variables. Similarly, we have that $\mathbf{u}^\top\mathbf{A}$ is in the left kernel of $\mathbf{C}_1$ for all vectors $\mathbf{u}$ in the left kernel of $\mathbf{D}_1$. We obtain another $r(n-r)$ linear equations in the $\mathbf{A}$ variables. Then, we can apply the same hybrid approach as in the two-collision case and we can further use the improved modeling from Section 4.3.1. The cost of solving this system, denoted by $\mathsf{C}_{\mathsf{solve}}$, follows directly from the analysis in Section 4.3.1, with the exception that we have fewer variables because the collision allows us to express $\mathbf{A}$ and $\mathbf{B}$ as spaces of dimension $n^2 - r(n-r)$. We need to solve this system for each candidate collision, so the overall complexity for a target rank r is

$$\mathcal{O}(\max(q^{\frac{\sigma}{2}}\mathsf{MR}_{(n,n,n-\sigma,r)}, q^\sigma \mathsf{C}_{\mathsf{solve}})).$$

In the quantum setting, we can use Grover's algorithm to speed up the creation of the lists of low-rank codewords, as well as the collision search. Hence, the quantum complexity of the attack is

$$\mathcal{O}(\mathsf{max}(q^{\frac{\sigma}{4}}\mathsf{MR}_{(n,n,n-\sigma,r)}, q^{\frac{\sigma}{2}}\mathsf{C}_{\mathsf{solve}})).$$

4.3.3 Graph-Theoretic Algorithm for Solving Polynomial Equivalence

This algorithm is an application of the graph-theoretic algorithms for solving the Quadratic Maps Linear Equivalence (QMLE) [BFV13] on MCE. This is possible since MCE is polynomial-time equivalent to the homogeneous version of QMLE [RST22]. Specifically, two matrix codes $\mathcal{C}$ and $\mathcal{D}$ from an MCE instance correspond to two quadratic maps $\mathcal{F}$ and $\mathcal{P}$ composed of k multivariate polynomials in $m+n$ variables. The state-of-the-art algorithm is a collision-search algorithm in two steps: First, create lists $L_{\mathcal{F}}$ and $L_{\mathcal{P}}$ of elements in $\mathbb{F}_q^{n+m}$ that satisfy a certain property $\mathbb{P}$ with regards to $\mathcal{F}$ and $\mathcal{P}$. Then, find a collision between these $L_{\mathcal{F}}$ and $L_{\mathcal{P}}$ that reduces to a much simpler equivalence problem that we can solve in polynomial time. The distinguishing property $\mathbb{P}$ can be any invariant of quadratic maps. Specifically, in [RST22] the authors propose to use the dimension of the kernel of the differential[3] associated to $\mathcal{F}$ resp. $\mathcal{P}$ for the case where $m+n \leqslant k \leqslant 2(m+n)$, and the zeros of $\mathcal{F}$ resp. $\mathcal{P}$ for the case $n \leqslant k \leqslant m+n$. The concrete complexity of the algorithm follows a birthday argument and is the maximum of the complexity of the two steps described above. Using the dimension of the kernel of the differential as a distinguishing property, the complexity of the attack is

$$\mathcal{O}(\mathsf{max}((q^{m+n}/d)^{\frac{1}{2}}, q^{m+n}d)), \tag{5}$$

up to some polynomial (for most parameter ranges) factors, and with success probability of $\approx 63\%$. Here, d is the proportion of elements satisfying $\mathbb{P}$ and can be calculated as $d = 1/\mathcal{O}(q^{\kappa^2+\kappa(k-(m+n))})$, where κ is an integer chosen such that it minimizes equation (5). This attack does not perform well for our parameter sets, because in the case of $n = m = k$, it is better to look for collisions between codewords instead of transforming the codes into quadratic maps. This is done in the Leon-like algorithm described in Section 4.3.2, with some additional improvements.

In the quantum setting, Grover's algorithm can be used to speed up both steps of the attack. In the first step, we can use Grover's algorithm for generating the lists $L_{\mathcal{F}}$ and $L_{\mathcal{P}}$, and in the second step, we can use it for the collision search. Factoring in this speed-up gives us the following complexity

$$\mathcal{O}(\mathsf{max}((q^{m+n}/d)^{\frac{1}{4}}, (q^{m+n}d)^{\frac{1}{2}})).$$

4.3.4 On the Applicability of an Attack Against ATFE

Since MCE is known to be equivalent to the alternating trilinear form equivalence (ATFE) problem [GQT22, TDJ+22], it is crucial to explore whether attack techniques that have been developed for ATFE apply to MCE. Specifically, we analysed the ideas from Beullens' attack [Beu22] on ATFE and their impact in the MCE setting. This is a collision-search attack that uses the graph of alternating trilinear forms $\phi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. A vertex in the graph is an element from a projective space over $\mathbb{F}_q^n$, of all elements $\mathbf{v} \in \mathbb{F}_q^n$ up to a scalar $\alpha \in \mathbb{F}_q$. Let us denote by $\phi_{\mathbf{u},\mathbf{v}}$ the linear form obtained by fixing two arguments of a trilinear form ϕ to $(\mathbf{u}, \mathbf{v})$. Then, there is an edge $(\mathbf{u}, \mathbf{v})$ in the graph if $\mathbf{u} \neq \mathbf{v}$ $(\mathbf{u}, \mathbf{v})$ and $\phi_{\mathbf{u},\mathbf{v}} = 0$. The rank of a vector $\mathbf{u} \in \mathbb{F}_q^n$

[3]This was also the choice in the original proposal [BFV13].

(with respect to ϕ) is defined as the number of vectors $\mathbf{v} \in \mathbb{F}_q^n$ such that $\phi_{\mathbf{u},\mathbf{v}} = 0$. Note that this is equivalent to the degree of a vertex $\mathbf{u}$. The graph defined this way is an invariant of trilinear forms and so, it can be used as a tool in a collision-search algorithm. The main idea of the algorithm is to look for collisions between low-rank elements. Finding low-rank elements is modeled as a MinRank problem and solved using state-of-the-art MinRank solving techniques.

If a matrix code is viewed as a (non-alternating) trilinear form, then the graph-theoretic approach described above corresponds to finding a collision between low-rank codewords. It is thus very similar to the Leon-like algorithm described in Section 4.3.2. However, the graph-theoretic approach from [Beu22] allows for a *graph walking* improvement that significantly reduces the complexity of the attack in the ATFE setting. When one low-rank element $\mathbf{u}$ is found, we continue looking for other low-rank elements in the neighbourhood of $\mathbf{u}$. This yields a smaller MinRank instance, because the neighbours of an element of rank r form a space of dimension $n - r$, so the corresponding MinRank is an instance of only $n - r$ matrices and a target rank r.

We applied this technique in the MCE setting, but no significant improvement was found, especially not when compared to the similar technique from Section 4.3.2. This is mainly because trilinear forms obtained from matrix codes are less structured than alternating trilinear forms. For instance, when an ATFE is transformed into a matrix code, all codewords are represented by anti-symmetric matrices, and consequently, they are all of even rank. Our theoretical analysis, as well as our experimental findings, show that there is no advantage in looking for low-rank codewords in the neighbourhood of a low-rank codeword instead of in the entire graph. For our parameter sets and the optimal target rank in the Leon-like algorithm, the probability that a low-rank codeword has low-rank neighbours is so small that we can not take advantage of graph walking to reduce the complexity of the attack.

5 Instantiation and Implementation

5.1 Parameter Sets (2.B.1 cont.)

The MEDS protocol has a flexible structure that accommodates a diverse range of parameter choices, allowing users to tailor the scheme to suit specific applications. The field size q, as well as the matrix code dimensions n, m, and k, are the main security parameters for the underlying matrix code equivalence problem, while the parameters s, t, and w control the security of the Fiat-Shamir construction.

All q, n, m, and k have a direct impact on the sizes of keys and signatures. The cost of some attacks (as discussed in Section 4) is defined by the minimum of n, m, and k; therefore, we are using $n = m = k$ in all proposed parameter sets. However, variances in the concrete attack cost provide some trade-offs between the selection of q and $n = m = k$; we are choosing these parameters such that we get the smallest key and signature sizes for a given security level.

There is also a trade-off between s, t, and w. The parameter s has a direct linear impact on public key size and key generation time, but a larger s allows us to chose smaller t and w. Parameter t has a linear effect on signing and verification time and w has a quasi-linear effect on the signature size.

For this submission, we decided to propose 'balanced' parameter sets where public key size and signature size are similar. We hence pick s such that a moderate signing and verification time can be achieved and then chose t and w such that the signature size is similar to the public key size and such that signing and verification time indeed are moderate.

Parameter Set	q	n	m	k	s	t	w	pk (byte)	sig (byte)	FS
					NIST Security Level I					
MEDS-9923	4 093	14	14	14	4	1 152	14	9 923	9 896	−128.11
MEDS-13220	4 093	14	14	14	5	192	20	13 220	12 976	−129.14
					NIST Security Level III					
MEDS-41711	4 093	22	22	22	4	608	26	41 711	41 080	−192.49
MEDS-69497	4 093	22	22	22	5	160	36	55 604	54 736	−191.34
					NIST Security Level V					
MEDS-134180	2 039	30	30	30	5	192	52	134 180	132 528	−261.84
MEDS-167717	2 039	30	30	30	6	112	66	167 717	165 464	−258.95

Table 2: MEDS parameter sets. (FS denotes the security of the Fiat-Shamir construction.)

Parameter Set	q	n	m	k	Algebraic Classical	Leon-like Classical	Leon-like Quantum
		NIST Security Level I					
MEDS-9923, MEDS-13220	4093	14	14	14	148.10	170.68	146.68
		NIST Security Level III					
MEDS-41711, MEDS-69497	4093	22	22	22	218.41	246.95	216.95
		NIST Security Level V					
MEDS-134180, MEDS-167717	2039	30	30	30	298.82	297.77	275.78

Table 3: Cost of the best known attacks for all parameter sets in log scale. The algebraic attack does not benefit from Grover's algorithm for these parameter sizes, so the complexity is the same in the classical and quantum setting.

Table 2 shows the resulting parameter sets for NIST security Levels I, III, and V. For each security level, we propose two parameter sets, a small one with small public key and signature sizes and a fast one with smaller t and hence faster signing and verification times. Different parameter sets for different trade-offs can be specified as described above.

The cost of the two best known attacks for our parameter sets, the algebraic attack described in Section 4.3.1 and the Leon-like attack from Section 4.3.2, is shown in Table 3.

The byte length of seeds and resulting key and signature sizes as well as the instantiations of XOF, H, and G are shown in Table 4.

5.2 Implementation Notes

Constant time implementations are considered the base-line for the secure implementation of cryptographic schemes. Besides a constant-time implementation of the basic finite field operations in $\mathbb{F}_q$ (addition, multiplication, modulo-q computation, inversion), the most crucial operations in MEDS are matrix multiplication, and a matrix systemizer for the computation of a systematic form. Matrix inversion can easily be implemented using the systemizer.

		Security Level		
		Level I	Level III	Level V
$\ell_{\mathsf{tree\_seed}}$	(byte)	16	24	32
$\ell_{\mathsf{sec\_seed}}$	(byte)	32	32	32
$\ell_{\mathsf{pub\_seed}}$	(byte)	32	32	32
ℓ_{salt}	(byte)	32	32	32
ℓ_{digest}	(byte)	32	32	32
$\ell_{\mathbb{F}_q^{m\times m}}$	(byte)	$\lceil m^2 \cdot \log_2(q)/8 \rceil$		
$\ell_{\mathbb{F}_q^{n\times n}}$	(byte)	$\lceil n^2 \cdot \log_2(q)/8 \rceil$		
$\ell_{\mathbf{G}_i}$	(byte)	$\lceil ((k-2)(mn-k)+n)\lceil \log_2(q) \rceil)/8 \rceil$		
ℓ_{sk}	(byte)	$(s-1)(\ell_{\mathbb{F}_q^{m\times m}} + \ell_{\mathbb{F}_q^{n\times n}}) + \ell_{\mathsf{sec\_seed}} + \ell_{\mathsf{pub\_seed}}$		
ℓ_{pk}	(byte)	$(s-1)\ell_{\mathbf{G}_i} + \ell_{\mathsf{pub\_seed}}$		
ℓ_{path}	(byte)	$(2^{\lceil \log_2(w) \rceil} + w(\lceil \log_2(t) \rceil - \lceil \log_2(w) \rceil - 1)) \cdot \ell_{\mathsf{tree\_seed}}$		
ℓ_{sig}	(byte)	$\ell_{\mathsf{digest}} + w(\ell_{\mathbb{F}_q^{m\times m}} + \ell_{\mathbb{F}_q^{n\times n}}) + \ell_{\mathsf{path}} + \ell_{\mathsf{salt}}$		
XOF		SHAKE256		
H		SHAKE256		
G		SHAKE256		

Table 4: Choice of functions and parameters.

In the reference implementation, we simply use schoolbook multiplication for the implementation of matrix multiplication, which does not have any data-dependent branches or address calculations. However, if other algorithms are used for the optimization of matrix multiplication, constant-time aspects need to be considered.

Our reference implementation is using a constant-time version of Gaussian elimination for computing the systematic form of a matrix (respective for matrix inversion), which results in a slightly higher computational cost than a straight-forward implementation of Gaussian elimination. The requirement of the systematic form for the pivot elements to be in the (left) diagonal, however, simplifies the constant-time implementation and reduces its cost compared to a general constant-time Gaussian elimination implementation.

System solving is a crucial operation in the key generation of MEDS. To implement the function Solve, we need to solve a linear system derived from:

$$\mathbf{P}_0\mathbf{B}^{-1} = \mathbf{A}\mathbf{P}_0^{(0)} \tag{6a}$$

$$\mathbf{P}_1\mathbf{B}^{-1} = \mathbf{A}\mathbf{P}_1^{(0)} \tag{6b}$$

with $\mathbf{P}_0 = \mathbf{I}_m$ and $\mathbf{P}_1 = \mathbf{U}_m$ for $\mathbf{A} \in \mathbb{F}_q^{m\times m}$ and $\mathbf{B}^{-1} \in \mathbb{F}_q^{n\times n}$ in $n^2 + m^2 - 1$ variables and $2mn - 1$ equations, which in a straight-forward implementation can lead to a significant computational cost for key generation in particular for larger parameter sets. However, due to the sparse structure of $\mathbf{P}_0$ and $\mathbf{P}_1$, the resulting system is sparse and highly structured. The cost of solving the system hence can be reduced to that of solving a $\mathbb{F}_q^{n\times(2m+2)}$ system derived from (6a) and (6b) plus back-substitution for all $n^2 + m^2 - 1$ variables. See the reference implementation for further details. For $n = m$ as in our parameter sets, the resulting cost is in the order of computing the inverse of a $\mathbb{F}_q^{n\times n}$ matrix. We are using our constant-time Gaussian elimination implementation to solve the derived smaller system in the reference implementation.

Parameter Set	Key Gen.		Signing		Verification		pk	sk	sig
	(ms)	(mcyc.)	(ms)	(mcyc.)	(ms)	(mcyc.)	(byte)	(byte)	(byte)
NIST Security Level I									
MEDS-9923	1.00	1.90	272.66	518.05	271.36	515.58	9 923	1 828	9 896
MEDS-13220	1.32	2.51	46.79	88.90	46.04	87.48	13 220	2 416	12 976
NIST Security Level III									
MEDS-41711	5.16	9.80	772.10	1 467.00	769.46	1 461.97	41 711	4 420	41 080
MEDS-55604	6.75	12.82	203.83	387.27	200.37	380.70	55 604	5 872	54 736
NIST Security Level V									
MEDS-134180	23.55	44.75	857.81	1 629.84	848.72	1 612.57	134 180	9 968	132 528
MEDS-167717	29.39	55.83	506.21	961.80	494.15	938.89	167 717	12 444	165 464

Table 5: Performance of the reference implementation of MEDS.

Note: Since we use only the first $2mn - 1$ equations of the $2mn$ equations provided by (6a) and (6b) by dropping one equation from (6b), the complete relation in (6b) does not hold and the rows in the public key matrices $\mathbf{G}_i$ corresponding to $\mathbf{P}_1$ actually have non-zero entries corresponding to the last row of $\mathbf{P}_1$. See Algorithm 3 on the storage of these entries.

5.3 Performance (2.B.2)

Table 5 shows the performance and data sizes of the reference implementation in time (ms) and megacycles (mcyc.) at 1 900 MHz on an AMD Ryzen 7 PRO 5850U CPU with 32GB RAM running under Ubuntu 22.04.1 Linux following the SUPERCOP setup[4] computed as median of 128 randomly seeded runs each.

The reference implementation of MEDS is also the optimized implementation since there is no particular advantage from using a 64-bit architecture for MEDS. However, due to the highly parallel nature of signing and verification (all t iterations can be computed in parallel), we expect a significant performance improvement from a parallel SIMD implementation.

The most recent version of the reference implementation is available on GitHub:

`https://github.com/MEDSpqc/meds`

6 Known Answer Tests (2.B.3)

We provide Known Answer Tests at the URL:

`https://www.meds-pqc.org/KAT/MEDS-KAT-2023-06-30.tgz`

7 Advantages and Limitations (2.B.6)

7.1 Advantages

Simplicity. Assuming the hardness of the MCE problem, MEDS is based on a well-known Fiat-Shamir construction. Overall, the scheme is very simple, given the building blocks of matrix arithmetic and matrix systemization. The most complex operation is the construction of the seed tree, for which several efficient and straightforward approaches exist.

[4]`https://bench.cr.yp.to/supercop.html`

Flexibility. With q and $n = m = k$ as well as s, t, and w, we have several parameters to control public key and signature sizes as well as computational cost within given security requirements. This enables us to optimize parameter sets for a given security level for small signatures, small public keys, or — as presented in our parameter set proposal — balanced public key and signature sizes at moderate computational cost.

7.2 Limitations

Large Data Size and Scalability. Despite the flexibility in the parameter selection, the resulting public keys and signatures are relatively large. This is even more significant when scaling up the parameters to higher parameter sets. However, Section 8 describes preliminary work that would allow us to significantly reduce signature sizes. We are currently evaluating this approach in regard to security, computational efficiency, and key- vs. signature-size trade-offs (e.g., using a smaller s), and we are considering incorporating such an approach into our proposal in the future, if the security analysis and the time-memory trade-offs turns out to be favorable. This approach then would also improve the scalability to higher parameter sets.

8 Considerations for Reducing the Signature Size

This section explains an idea that can be used to reduce the signature size by a large factor. The idea makes use of the technique that we use to reduce the public key size. The proposed parameter sets do not make use of the idea, but we might want to apply it for the second round. For simplicity, below we explain the idea for $s = 2$, although it is easy to see that the idea also works for any $s > 2$.

Let the public key be $(\mathbf{G}_0, \mathbf{G}_1)$ such that

$$\mathbf{G}_0 = \mathsf{SF}(\pi_{\mathbf{A}_1^{-1},\mathbf{B}_1^{-1}}(\mathbf{G}_1)) = \mathbf{T}_1^{-1} \cdot \pi_{\mathbf{A}_1^{-1},\mathbf{B}_1^{-1}}(\mathbf{G}_1) = \pi_{\mathbf{A}_1^{-1},\mathbf{B}_1^{-1}}(\mathbf{T}_1^{-1} \cdot \mathbf{G}_1).$$

To compute $\tilde{\mathbf{G}}_j$'s, the signer first generates a seed σ_r and expand it into full-rank matrices $\mathbf{R}_0, \dots, \mathbf{R}_{t-1} \in \mathbb{F}_q^{2\times mn}$. Then, for $j \in \{0, \dots, t-1\}$, the following steps are carried out.

- Generate a seed σ_j and expand it into a full-rank matrix $\mathbf{M}_j \in \mathbb{F}_q^{2\times k}$.
- From $\mathbf{M}_j \cdot \mathbf{G}_0 \in \mathbb{F}_q^{2\times mn}$ and $\mathbf{R}_j$, derive $\tilde{\mathbf{A}}_j, \tilde{\mathbf{B}}_j$ such that $\pi_{\tilde{\mathbf{A}}_j,\tilde{\mathbf{B}}_j}(\mathbf{M}_j \cdot \mathbf{G}_0) = \mathbf{R}_j$.
- Compute $\tilde{\mathbf{G}}_j = \mathsf{SF}(\pi_{\tilde{\mathbf{A}}_j,\tilde{\mathbf{B}}_j}(\mathbf{G}_0)) = \mathsf{SF}(\pi_{\tilde{\mathbf{A}}_j\cdot\mathbf{A}^{-1},\mathbf{B}^{-1}\cdot\tilde{\mathbf{B}}_j}(\mathbf{G}_1))$.

A signature will then include

- the seed σ_r,
- σ_j's for all j with $h_j = 0$,
- $\mathbf{M}_j \cdot \mathbf{T}_1^{-1} \in \mathbb{F}_q^{2\times k}$ for all j with $h_j = 1$, and
- the digest d, which is obtained by hashing $\tilde{\mathbf{G}}_j$'s and the message.

From σ_r and σ_j, the verifier can derive $(\tilde{\mathbf{A}}_j, \tilde{\mathbf{B}}_j)$ and thus $\tilde{\mathbf{G}}_j$ for all j with $h_j = 0$. What is less obvious is that from σ_r and $\mathbf{M}_j \cdot \mathbf{T}_1^{-1}$, the verifier can also derive $\tilde{\mathbf{G}}_j$ for all j with $h_j = 1$. To see this, note that

$$\pi_{\tilde{\mathbf{A}}_j,\tilde{\mathbf{B}}_j}(\mathbf{M}_j \cdot \mathbf{G}_0) = \pi_{\tilde{\mathbf{A}}_j\cdot\mathbf{A}^{-1},\mathbf{B}^{-1}\cdot\tilde{\mathbf{B}}_j}(\mathbf{M}_j \cdot \mathbf{T}_1^{-1} \cdot \mathbf{G}_1) = \mathbf{R}_j.$$

Therefore, from $\mathbf{M}_j \cdot \mathbf{T}_1^{-1} \cdot \mathbf{G}_1$ and $\mathbf{R}_j$, the verifier can derive $\tilde{\mathbf{A}}_j \cdot \mathbf{A}^{-1}, \mathbf{B}^{-1} \cdot \tilde{\mathbf{B}}_j$ and thus $\tilde{\mathbf{G}}_j$ for all j with $h_j = 1$.

When the idea is applied directly using the parameters for MEDS-13220, the signature size can be reduced from 12 916 bytes to only

$$32 + 16 \cdot (t - w) + \lceil 2 \cdot k \cdot \lceil \log_2(q) \rceil / 8 \rceil \cdot w + 32 = 3\,656$$

bytes. By replacing σ_j's by necessary seeds p in a hash tree and the corresponding salt α, the signature size can be further reduced to 2 088 bytes.

References

[BBC+20] Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray A. Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier A. Verbel. Improvements of algebraic attacks for solving the rank decoding and MinRank problems. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 507–536. Springer, Heidelberg, December 2020.

[BBPS21] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. LESS-FM: Fine-tuning signatures from the code equivalence problem. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, pages 23–43. Springer, Heidelberg, 2021.

[BCH+23] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. Oil and vinegar: Modern parameters and implementations. Cryptology ePrint Archive, Report 2023/059, 2023. https://eprint.iacr.org/2023/059.

[Beu19] Ward Beullens. Sigma protocols for MQ, PKP and SIS, and fishy signature schemes. Cryptology ePrint Archive, Report 2019/490, 2019. https://eprint.iacr.org/2019/490.

[Beu22] Ward Beullens. Graph-theoretic algorithms for the alternating trilinear form equivalence problem. Cryptology ePrint Archive, Report 2022/1528, 2022. https://eprint.iacr.org/2022/1528.

[BFV13] Charles Bouillaguet, Pierre-Alain Fouque, and Amandine Véber. Graph-theoretic algorithms for the "isomorphism of polynomials" problem. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 211–227. Springer, Heidelberg, May 2013.

[BKP20a] Ward Beullens, Shuichi Katsumata, and Federico Pintore. Calamari and Falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 464–492. Springer, Heidelberg, December 2020.

[BKP20b] Ward Beullens, Shuichi Katsumata, and Federico Pintore. Calamari and Falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. Cryptology ePrint Archive, Report 2020/646, 2020. https://eprint.iacr.org/2020/646.

[CDG20] Alain Couvreur, Thomas Debris-Alazard, and Philippe Gaborit. On the hardness of code equivalence problems in rank metric. arXiv:2011.04611 [cs.IT], 2020. https://arxiv.org/abs/2011.04611.

[CNP$^+$23] Tung Chou, Ruben Niederhagen, Edoardo Persichetti, Tovohery Hajatiana Randrianarisoa, Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. Take your MEDS: Digital Signatures from Matrix Code Equivalence. In *AFRICACRYPT 2023*, LNCS. Springer, 2023. To appear. Preprint: Cryptology ePrint Archive, Report 2022/1559, 2022. https://eprint.iacr.org/2022/1559.

[Cou01] Nicolas Courtois. Efficient zero-knowledge authentication based on a linear algebra problem MinRank. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 402–421. Springer, Heidelberg, December 2001.

[DCP$^+$20] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang, Matthias Kannwischer, and Jacques Patarin. Rainbow. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 356–383. Springer, Heidelberg, August 2019.

[DG19] Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 759–789. Springer, Heidelberg, May 2019.

[FLP08] Jean-Charles Faugère, Françoise Levy-dit-Vehel, and Ludovic Perret. Cryptanalysis of minrank. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 280–296. Springer, Heidelberg, August 2008.

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

[GQ23] Joshua Grochow and Youming Qiao. On the Complexity of Isomorphism Problems for Tensors, Groups, and Polynomials I: Tensor Isomorphism-Completeness. *SIAM Journal on Computing*, 52(2):568–617, 2023. Part of the preprint arXiv:1907.00309. Preliminary version appeared at ITCS '21, DOI:10.4230/LIPIcs.ITCS.2021.31.

[GQT22] Joshua A. Grochow, Youming Qiao, and Gang Tang. Average-case algorithms for testing isomorphism of polynomials, algebras, and multilinear forms. *Journal of Groups, Complexity, Cryptology*, Volume 14, Issue 1, August 2022. Preliminary version appeared in STACS '21, doi:10.4230/LIPIcs.STACS.2021.38. Preprint available at arXiv:2012.01085.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 552–586. Springer, Heidelberg, April / May 2018.

[KS99] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 19–30. Springer, Heidelberg, August 1999.

[Leo82] Jeffrey S. Leon. Computing automorphism groups of error-correcting codes. *IEEE Trans. Inf. Theory*, 28(3):496–510, 1982.

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting post-quantum Fiat-Shamir. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 326–355. Springer, Heidelberg, August 2019.

[Ran20] Robert Ransom. Constant-time verification for cut-and-choose-based signatures. Cryptology ePrint Archive, Report 2020/1184, 2020. https://eprint.iacr.org/2020/1184.

[RST22] Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. Hardness estimates of the code equivalence problem in the rank metric. Cryptology ePrint Archive, Report 2022/276, 2022. https://eprint.iacr.org/2022/276.

[TDJ+22] Gang Tang, Dung Hoang Duong, Antoine Joux, Thomas Plantard, Youming Qiao, and Willy Susilo. Practical post-quantum signature schemes from isomorphism problems of trilinear forms. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 582–612. Springer, Heidelberg, May / June 2022.

[Unr12] Dominique Unruh. Quantum proofs of knowledge. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 135–152. Springer, Heidelberg, April 2012.