Take Your MEDS: Digital Signatures from Matrix Code Equivalence

Tung Chou[1(✉)], Ruben Niederhagen[1,2(✉)], Edoardo Persichetti[3,4(✉)], Tovohery Hajatiana Randrianarisoa[5(✉)], Krijn Reijnders[6(✉)], Simona Samardjiska[6(✉)], and Monika Trimoska[6(✉)]

[1] Academia Sinica, Taipei, Taiwan
blueprint@crypto.tw, ruben@polycephaly.org
[2] University of Southern Denmark, Odense, Denmark
[3] Florida Atlantic University, Boca Raton, USA
epersichetti@fau.edu
[4] Sapienza University, Rome, Italy
[5] Umea University, Umea, Sweden
tovo@aims.ac.za
[6] Radboud Universiteit, Nijmegen, The Netherlands
{krijn,simonas,mtrimoska}@cs.ru.nl

Abstract. In this paper, we show how to use the Matrix Code Equivalence (MCE) problem as a new basis to construct signature schemes. This extends previous work on using isomorphism problems for signature schemes, a trend that has recently emerged in post-quantum cryptography. Our new formulation leverages a more general problem and allows for smaller data sizes, achieving competitive performance and great flexibility. Using MCE, we construct a zero-knowledge protocol which we turn into a signature scheme named Matrix Equivalence Digital Signature (MEDS). We provide an initial choice of parameters for MEDS, tailored to NIST's Category 1 security level, yielding public keys as small as 2.8 kB and signatures ranging from 18 kB to just around 6.5 kB, along with a reference implementation in C.

Keywords: group action · signature scheme · code-based cryptography · post-quantum cryptography · matrix codes

1 Introduction

Post-Quantum Cryptography (PQC) comprises all the primitives that are believed to be resistant against attackers equipped with a considerable quantum

An extended and correctly typeset version of this paper can be found at https://eprint.iacr.org/2022/1559.

Tung Chou is supported by Taiwan National Science and Technology Council (NSTC, previously Ministry of Science and Technology) grant 109-2222-E-001-001-MY3. Edoardo Persichetti is supported by NSF grant 1906360 and NSA grant H98230-22-1-0328. Monika Trimoska is supported by the ERC Starting Grant 805031 (EPOQUE).

N. El Mrabet et al. (Eds.): AFRICACRYPT 2023, LNCS 14064, pp. 28–52, 2023.
https://doi.org/10.1007/978-3-031-37679-5_2

computing power. Several such schemes have been around for a long time [39,44], some being in fact almost as old as RSA [36]; however, the area itself was not formalized as a whole until the early 2000s, for instance with the first edition of the PQCrypto conference [37]. The area has seen a dramatic increase in importance and volume of research over the past few years, partially thanks to NIST's interest and the launch of the PQC Standardization process in 2017 [38]. After 4 years and 3 rounds of evaluation, the process has crystallized certain mathematical tools as standard building blocks (e.g. lattices, linear codes, multivariate equations, isogenies etc.). Some algorithms [31,35,42] have now been selected for standardization, with an additional one or two to be selected among a restricted set of alternates [1,4,5] after another round of evaluation. While having a range of candidates ready for standardization may seem satisfactory, research is still active in designing PQC primitives. In particular, NIST has expressed the desire for a greater diversity among the hardness assumptions behind signature schemes, and announced a partial re-opening of the standardization process for precisely the purpose of collecting non-lattice-based protocols.

Cryptographic group actions are a popular and powerful instrument for constructing secure and efficient cryptographic protocols. The most well-known is, without a doubt, the action of finite groups on the integers modulo a prime, or the set of points on an elliptic curve, which give rise to the *Discrete Logarithm Problem (DLP)*, i.e. the backbone of public-key cryptography. Recently, proposals for post-quantum cryptographic group actions started to emerge, based on the tools identified above: for instance, isogenies [18], linear codes [14], trilinear forms [47] and even lattices [30]. All of these group actions provide very promising solutions for cryptographic schemes, for example signatures [8,23,47] and many others; at the same time, they are very different in nature, with unique positive and negative aspects.

Our Contribution. In this work, we formalize a new cryptographic group action based on the notion of *Matrix Code Equivalence*. This is similar in nature to the *code equivalence* notion at the basis of LESS [8,14], and in fact belongs to a larger class of isomorphism problems that include, for example, the lattice isomorphism problem, and the well-known isomorphism of polynomials [39]. The hardness of the MCE problem was studied in [22,45], from which it is possible to conclude that this is a suitable problem for post-quantum cryptography. Indeed, we show that it is possible to use MCE to build a zero-knowledge protocol, and hence a signature scheme, which we name *Matrix Equivalence Digital Signature*, or simply MEDS. For our security analysis, we first study in detail the collision attacks from [45] and then we develop two new attacks. The first attack that we propose uses a nontrivial algebraic modeling inspired from the minors modellings of MinRank in [7,26]. The second one is an adaptation of Leon's algorithm [34] for matrix codes. Based on this analysis, we provide an initial parameter choice, together with several computational optimizations, resulting in a scheme with great flexibility and very competitive data sizes. This group action allows for the construction of (linkable) ring signatures, with performance results that improve on the existing state of the art [9]. Due to limitations in space, the construction of ring signatures is included in an extended version of this work.

2 Preliminaries

Let $\mathbb{F}_q$ be the finite field of q elements. $\mathrm{GL}_n(q)$ and $\mathrm{AGL}_n(q)$ denote respectively the general linear group and the general affine group of degree n over $\mathbb{F}_q$. We use bold letters to denote vectors $\mathbf{a}, \mathbf{c}, \mathbf{x}, \ldots$, and matrices $\mathbf{A}, \mathbf{B}, \ldots$. The entries of a vector $\mathbf{a}$ are denoted by a_i, and we write $\mathbf{a} = (a_1, \ldots, a_n)$ for a (row) vector of dimension n over some field. Similarly, the entries of a matrix $\mathbf{A}$ are denoted by a_{ij}. Random sampling from a set S is denoted by $a \xleftarrow{\$} S$. For two matrices $\mathbf{A}$ and $\mathbf{B}$, we denote the Kronecker product by $\mathbf{A} \otimes \mathbf{B}$. Finally, we denote the set of all $m \times n$ matrices over $\mathbb{F}_q$ by $\mathcal{M}_{m,n}(\mathbb{F}_q)$.

2.1 Cryptographic Group Actions

Definition 1. Let X be a set and $(G, \cdot)$ be a group. A group action is a mapping

$$\begin{aligned} \star : G \times X &\to X \\ (g, x) &\mapsto g \star x \end{aligned}$$

such that the following conditions hold for all $x \in X$:

- $e \star x = x$, where e is the identity element of G.
- $g_2 \star (g_1 \star x) = (g_2 \cdot g_1) \star x$, for all $g_1, g_2 \in G$.

A group action can have a number of mathematically desirable properties. For example, we say that a group action is:

- *Commutative*: for any $g_1, g_2 \in G$, we have $g_2 \star (g_1 \star x) = g_1 \star (g_2 \star x)$.
- *Transitive*: given $x_1, x_2 \in X$, there is some $g \in G$ such that $g \star x_1 = x_2$.
- *Free*: if $g \star x = x$, then g is the identity.

In particular, a *cryptographic* group action is a group action with some additional properties that are useful for cryptographic applications. To begin with, there are some desirable properties of computational nature. Namely, the following procedures should be efficient:

- *Evaluation*: given x and g, compute $g \star x$.
- *Sampling*: sample uniformly at random from G.
- *Membership testing*: verify that $x \in X$.

Finally, cryptographic group actions should come with security guarantees; for instance, the *vectorization problem* should be hard:

Problem 1 (Group Action Vectorization).
Given: The pair $x_1, x_2 \in X$.
Goal: Find, if any, $g \in G$ such that $g \star x_1 = x_2$.

Early constructions using this paradigm are based on the action of finite groups of prime order, for which the vectorization problem is the discrete logarithm problem. Lately, multiple isogeny-based constructions have appeared: see, for instance, the work of Couveignes in [21] and later by Rostovtsev and Stolbunov [46]. A general framework based on group actions was explored in more detail by [3], allowing for the design of several primitives. The holy grail are those cryptographic group actions that possess both the mathematical and cryptographic properties listed above. Currently, CSIDH [18] is the only post-quantum commutative cryptographic group action, although there is an ongoing debate about the efficiency and quantum hardness of its vectorization problem [15]. In Sect. 3, we introduce the group action that is relevant to our work.

2.2 Protocols

We give here an explicit characterization of the protocols we will build. The corresponding security definitions are presented only in an informal manner; formal definitions will are included in the full version of this work.[1]

Definition 2. A *Sigma protocol* is a three-pass interactive protocol between two parties: a prover $\mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2)$ and a verifier $\mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2)$. The protocol is composed of the following procedures:

I. Keygen: on input some public data (including system parameters), output a public key pk (the instance) and the corresponding secret key sk (the witness). Give sk to the prover; pk is distributed publicly and is available to all parties. For simplicity, we assume that the public data is available as input in all the remaining procedures.
II. Commit: on input the public key pk, P_1 outputs a public commitment cmt and sends it to the verifier.
III. Challenge: on input the public key pk and the commitment cmt, V_1 samples uniformly at random a challenge ch from the challenge space C and sends it to the prover.
IV. Response: on input the secret key sk, the public key pk, the commitment cmt and the challenge ch, P_2 outputs a response rsp and sends it to the verifier.
V. Verify: on input a public key pk, the commitment cmt, the challenge ch, and the response rsp, V_2 outputs either 1 (accept) if the *transcript* $(\mathsf{cmt}, \mathsf{ch}, \mathsf{rsp})$ is valid, or 0 (reject) otherwise.

A Sigma protocol is usually required to satisfy the following properties. First, if the statement is true, an honest prover is always able to convince an honest verifier. This property is called *Completeness*. Secondly, a dishonest prover cannot convince an honest verifier other than with a small probability. This is captured by the *Soundness* property, which also bounds such probability, usually known as *soundness error* or, informally, *cheating probability*. Finally, the protocol has to be *Zero-Knowledge*, i.e. anyone observing the transcript (including the verifier) learns nothing other than the fact that the statement is true.

[1] https://eprint.iacr.org/2022/1559.pdf.

Definition 3. A *Digital Signature scheme* is a protocol between 2 parties: a signer S and a verifier V. The protocol is composed of the following procedures:

I. Keygen: on input the public data (including system parameters), output a secret signing key sk for S and the corresponding public verification key pk.
II. Sign: on input a secret key sk and a message msg, output a signature σ.
III. Verify: on input a public key pk, a message msg and a signature σ, V outputs either 1 (accept) if the signature is valid, or 0 (reject) otherwise.

Correctness means that an honest signer is always able to get verified. The usual desired security notion for signature schemes is *Unforgeability*, which guarantees computationally infeasible to forge a valid signature without knowing the secret signing key. Again, we leave formal definitions to the full version of this work.

3 The Matrix Code Equivalence Problem

A $[m \times n, k]$ *matrix code* is a subspace $\mathcal{C}$ of $\mathcal{M}_{m,n}(\mathbb{F}_q)$. These objects are usually measured with the rank metric, where the *distance* between two matrices $\mathbf{A}, \mathbf{B} \in \mathcal{M}_{m,n}(\mathbb{F}_q)$ is defined as $d(\mathbf{A}, \mathbf{B}) = \mathrm{Rank}(\mathbf{A} - \mathbf{B})$. We denote the basis of the subspace by $\langle \mathbf{C_1}, \ldots, \mathbf{C_k} \rangle$, where the $\mathbf{C_i}$'s are linearly independent elements of $\mathcal{M}_{m,n}(\mathbb{F}_q)$. Due to symmetry, without loss of generality, in the rest of the text we will assume $m \leqslant n$.

For a matrix $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{F}_q)$, let vec be a mapping that sends a matrix $\mathbf{A}$ to the vector $\mathsf{vec}(\mathbf{A}) \in \mathbb{F}_q^{mn}$ obtained by 'flattening' $\mathbf{A}$, i.e.:

$$\mathsf{vec}: \ \mathbf{A} = \begin{pmatrix} a_{1,1} & \ldots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \ldots & a_{m,n} \end{pmatrix} \mapsto \mathsf{vec}(\mathbf{A}) = (a_{1,1}, \ldots, a_{1,n}, \ldots, a_{m,1}, \ldots, a_{m,n}).$$

The inverse operation is denoted by mat, i.e. $\mathsf{mat}(\mathsf{vec}(\mathbf{A})) = \mathbf{A}$. Using the map vec, an $[m \times n, k]$ matrix code can be thought of as an $\mathbb{F}_q$-subspace of $\mathbb{F}_q^{mn}$, and thus we can represent it with a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times mn}$, in a manner similar to the common representation for linear codes. Indeed, if $\mathcal{C}$ is an $[m \times n, k]$ matrix code over $\mathbb{F}_q$, we denote by $\mathsf{vec}(\mathcal{C})$ the vectorization of $\mathcal{C}$ i.e.:

$$\mathsf{vec}(\mathcal{C}) := \{\mathsf{vec}(\mathbf{A}) \colon \mathbf{A} \in \mathcal{C}\}.$$

In this case, $\mathsf{vec}(\mathcal{C})$ is a k-dimensional $\mathbb{F}_q$-subspace of $\mathbb{F}_q^{mn}$.

Definition 4. Let $\mathcal{C}$ and $\mathcal{D}$ be two $[m \times n, k]$ matrix codes over $\mathbb{F}_q$. We say that $\mathcal{C}$ and $\mathcal{D}$ are *equivalent* if there exist two matrices $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$ such that $\mathcal{D} = \mathbf{A}\mathcal{C}\mathbf{B}$, i.e. for all $\mathbf{C} \in \mathcal{C}$, $\mathbf{ACB} \in \mathcal{D}$.

The equivalence between two matrix codes can be expressed using the Kronecker product of $\mathbf{A}^\top$ and $\mathbf{B}$, which we denote by $\mathbf{A}^\top \otimes \mathbf{B}$.

Lemma 1. *Let $\mathcal{C}$ and $\mathcal{D}$ be two $[m \times n, k]$ matrix codes over $\mathbb{F}_q$. Suppose that $\mathcal{C}$ and $\mathcal{D}$ are equivalent with $\mathcal{D} = \mathbf{A}\mathcal{C}\mathbf{B}$, with $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$. If $\mathbf{G}$ and $\mathbf{G}'$ are generator matrices for $\mathcal{C}$ and $\mathcal{D}$ respectively, then there exists a $\mathbf{T} \in \mathrm{GL}_k(q)$ such that $\mathbf{G}' = \mathbf{T}\mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B})$.*

It is common to write the generator matrices in systematic form (i.e., as a matrix of the shape $(I|M)$); we denote this operation by SF. Following Lemma 1, this gives us that $\mathcal{D} = \mathbf{A}\mathcal{C}\mathbf{B}$ if and only if $\mathsf{SF}(\mathbf{G}') = \mathsf{SF}(\mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B}))$.
To simplify notation, we introduce the following operator:

$$\pi_{\mathbf{A},\mathbf{B}}(\mathbf{G}) := \mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B}).$$

We are now ready to describe some hard problems connected to the objects we just introduced. The Matrix Code Equivalence (MCE) problem is formally defined as follows:

Problem 2 (Matrix Code Equivalence).
$\mathsf{MCE}(k, n, m, \mathcal{C}, \mathcal{D})$:
Given: Two k-dimensional matrix codes $\mathcal{C}, \mathcal{D} \subset \mathcal{M}_{m,n}(q)$.
Goal: Determine if there exist $\mathbf{A} \in \mathrm{GL}_m(q), \mathbf{B} \in \mathrm{GL}_n(q)$ such that $\mathcal{D} = \mathbf{A}\mathcal{C}\mathbf{B}$.

The map $(\mathbf{A}, \mathbf{B}) : \mathbf{C} \mapsto \mathbf{ACB}$ is an *isometry* between $\mathcal{C}$ and $\mathcal{D}$, in the sense that it preserves the rank i.e. $\mathrm{Rank}\,\mathbf{C} = \mathrm{Rank}(\mathbf{ACB})$. When $n = m$, such isometries can also be extended by transpositions of codewords, however, we choose to work with this smaller set of isometries for simplicity, at no cost to cryptographic security. Note that, although we defined MCE as a decisional problem, our signature construction relies on the computational version of it.

Remark 1. We thank Giuseppe D'Alconzo for the following sharp observation: An MCE instance of dimension k with $m \times n$ matrices over $\mathbb{F}_q$ can be viewed as a 3-tensor problem, which is symmetrical in its arguments k, m and n. This means that it is equivalent to an MCE instance of dimension m with $k \times n$ matrices and to an MCE instance of dimension n with $k \times m$ matrices. Switching to equivalent instances is a matter of changing perspective on the $k \times m \times n$ object over $\mathbb{F}_q$ defined by $A_{ijl} = A_{ij}^{(l)}$. In other words, each basis matrix $m \times n$ defines a slice of a cube, and one can take different slices for equivalent instances.

Finally, we present a *multiple-instance* version of MCE, which is at the base of one of the optimizations, using *multiple public keys*, which we will describe in Sect. 5. It is easy to see that this new problem reduces to MCE, as done for instance in [8] for the Hamming case.

Problem 3 (Multiple Matrix Code Equivalence).
$\mathsf{MMCE}(k, n, m, r, \mathcal{C}, \mathcal{D}_1, \ldots, \mathcal{D}_r)$:
Given: $(r+1)$ k-dimensional matrix codes $\mathcal{C}, \mathcal{D}_1, \ldots, \mathcal{D}_r \subset \mathcal{M}_{m,n}(\mathbb{F}_q)$.
Goal: Find – if any – $\mathbf{A} \in \mathrm{GL}_m(q), \mathbf{B} \in \mathrm{GL}_n(q)$ such that $\mathcal{D}_i = \mathbf{A}\mathcal{C}\mathbf{B}$ for some $i \in \{1, \ldots, r\}$.

The MCE problem has been shown to be at least as hard as the Code Equivalence problem in the Hamming metric [22]. Furthermore, under moderate assumptions, MCE is equivalent to the homogeneous version of the Quadratic Maps Linear Equivalence problem (QMLE) [45], which is considered the hardest among polynomial equivalence problems. An extensive security evaluation will be given in Sect. 6, encompassing an overview of the best attack techniques and concrete security estimates. From this, we infer a choice of parameters in Sect. 7.1.

To conclude, we now lay out the details of the MCE-based group action, given by the action of isometries on k-dimensional matrix codes. That is, the set X is formed by the k-dimensional matrix codes of size $m \times n$ over some base field $\mathbb{F}_q$, and the group $G = \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$ acts on this set via isometries as follows:

$$\begin{aligned} \star : \quad G \times X \quad &\to \quad X \\ ((\mathbf{A}, \mathbf{B}), \mathcal{C}) &\mapsto \mathbf{A}\mathcal{C}\mathbf{B} \end{aligned}$$

We write $\mathcal{G}_{m,n}(q)$ to denote this group of isometries and $\mathcal{M}_{k,m,n}(q)$ for the set of k-dimensional matrix codes; to simplify notation, we drop the indices k, m, n and q when clear from context. Then, for this MCE-based group action the Vectorization Problem is precisely Problem 2. This action is not commutative and in general neither transitive nor free. We can restrict the set $\mathcal{M}$ to a single well-chosen orbit to make the group action both transitive and free. In fact, picking any orbit generated from some starting code $\mathcal{C}$ ensures transitivity, and the group action is free if the chosen code $\mathcal{C}$ has trivial automorphism group $\mathrm{Aut}_{\mathcal{G}}(\mathcal{C}) := \{\varphi \in \mathcal{G} : \varphi(\mathcal{C}) = \mathcal{C}\}$, where trivial means up to scalars in $\mathbb{F}_q$[2]. The non-commutativity is both positive and negative: although it limits the cryptographical design possibilities, e.g. key exchange becomes hard, it prevents quantum attacks to which commutative cryptographic group actions are vulnerable, such as Kuperberg's algorithm for the dihedral hidden subgroup problem [33].

With regards to efficiency, it is immediate to notice that our group action is very promising, given that the entirety of the operations in the proposed protocols is simple linear algebra; this is in contrast with code-based literature (where complex decoding algorithms are usually required) and other group actions (e.g. isogeny-based) which are burdened by computationally heavy operations. Further details about performance are given in details about performance are given Sect. 7.

4 Protocols from Matrix Code Equivalence

The efficient non-commutative cryptographic group action provided by MCE from Sect. 3 yields a promising building block for post-quantum cryptographic schemes. In this section, we obtain a digital signature scheme by

[2] More accurately, as the action of an isometry (A, B) is only interesting up to scalars $\lambda, \mu \in \mathbb{F}_q$, the group that is acting *freely* is $\mathrm{PGL}_m(q) \times \mathrm{PGL}_n(q)$.

Public Data

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

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 $h = \mathsf{hash}(\tilde{\mathbf{G}})$.
4. Set $\mathsf{cmt} = h$.
5. Send cmt to verifier.

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.

I. Keygen()

1. $\mathbf{G}_0 \xleftarrow{\$} \mathbb{F}_q^{k \times mn}$ in standard form
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 $\mathsf{pk} = (\mathbf{G}_0, \mathbf{G}_1)$.

III. Challenge()

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

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

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

Fig. 1. MCE Sigma Protocol

first designing a Sigma protocol and then applying the Fiat-Shamir transformation [28].

The first building block in our work is the Sigma protocol in Fig. 1, in which a Prover proves the knowledge of an isometry $(\mathbf{A}, \mathbf{B})$ between two equivalent matrix codes. The security result is given in Theorem 1. The proof is considered standard in the literature (similar to the one given in [14], for instance) and is therefore omitted in the interest of space.

Theorem 1. *The Sigma protocol described above is complete, 2-special sound and honest-verifier zero-knowledge assuming the hardness of the* MCE *problem.*

Applying the Fiat-Shamir transformation gives the signature scheme in Fig. 2.

Public Key and Signature Size. We begin by calculating the communication costs for the Sigma protocol of Fig. 1. Note that, for the case $c = 0$, the response (μ, ν) consists entirely of randomly-generated objects, and is efficiently represented by a single seed (that can be used to generate both matrices). This yields the following cost per round, in bits:

$$\begin{cases} 3\lambda + 1 & \text{if } c = 0 \\ 2\lambda + 1 + (m^2 + n^2)\lceil \log_2(q) \rceil & \text{if } c = 1 \end{cases}$$

remembering that seeds are λ bits and hash digests 2λ to avoid collision attacks.

For the signature scheme we calculate the sizes as follows. First, since the matrix $\mathbf{G}_0$ is random, it can also be represented via a short seed, and therefore

Public Data

$q, m, n, k, t = \lambda \in \mathbb{N}$.
$\mathsf{hash} : \{0,1\}^* \to \{0,1\}^t$.

II. Sign(sk)

1. For all $i = 0, \ldots, t-1$:
   i. $\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
   ii. Compute $\tilde{\mathbf{G}}_i = \mathsf{SF}(\pi_{\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i}(\mathbf{G}_0))$.
2. Compute $h = \mathsf{hash}(\tilde{\mathbf{G}}_0, \ldots, \tilde{\mathbf{G}}_{t-1}, \mathsf{msg})$.
3. Parse $h = [h_0 | \ldots | h_{t-1}]$, $h_i \in \{0,1\}$.
4. For all $i = 0, \ldots, t-1$:
   i. Set $(\mu_i, \nu_i) = (\tilde{\mathbf{A}}_i \mathbf{A}_{h_i}^{-1}, \mathbf{B}_{h_i}^{-1} \tilde{\mathbf{B}}_i)$.
5. Set $\sigma = (h, \mu_0, \ldots, \mu_{t-1}, \nu_0, \ldots, \nu_{t-1})$.
6. Send σ to verifier.

I. Keygen()

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

III. Verify(pk, msg, σ)

1. Parse $h = [h_0 | \ldots | h_{t-1}]$, $h_i \in \{0,1\}$.
2. For all $i = 0, \ldots, t-1$:
   i. Set $\hat{\mathbf{G}}_i = \mathsf{SF}(\pi_{\mu_i, \nu_i}(\mathbf{G}_{h_i}))$.
3. Compute $h' = \mathsf{hash}(\hat{\mathbf{G}}_0, \ldots, \hat{\mathbf{G}}_{t-1}, \mathsf{msg})$.
4. Accept if $h' = h$ or reject otherwise.

Fig. 2. The basic signature scheme

can be included in the public key at negligible cost (see Algorithm I. of Fig. 2). Keeping in mind that the number of rounds t is equal to the value of the desired security level λ, the protocol above yields the following sizes (in bits):

- Public key size: $\lambda + k(mn-k)\lceil \log_2(q) \rceil$
- Average signature size: $t\left(1 + \dfrac{\lambda + (m^2+n^2)\lceil \log_2(q) \rceil}{2}\right)$.

5 Matrix Equivalence Digital Signature—MEDS

We apply the following optimizations from the literature to the basic Fiat-Shamir-based signature scheme described in Sect. 4, to obtain our Matrix Equivalence Digital Signature (MEDS).

Multiple Keys. The first optimization is a popular one in literature [8,13,23], and it consists of utilizing multiple public keys, i.e. multiple equivalent codes $\mathbf{G}_0, \ldots, \mathbf{G}_{s-1}$, each defined as $\mathbf{G}_i = \mathsf{SF}(\pi_{\mathbf{A}_i, \mathbf{B}_i}(\mathbf{G}_0))$ for uniformly chosen secret keys[3] $(\mathbf{A}_i, \mathbf{B}_i)$. This allows to reduce the soundness error from $1/2$ to $1/2^\ell$, where $\ell = \lceil \log_2 s \rceil$. The optimization works by grouping the challenge bits into strings of ℓ bits, which can then be interpreted as binary representations of the indices $\{0, \ldots, s-1\}$, thus dictating which public key will be used in the protocol. Security is preserved since the proof of unforgeability can be easily modified to rely on a multi-instance version of the underlying problem: in our case, MMCE (Problem 3). Note that, although in the literature s is chosen to be a power of 2, this does not have to be the case. In this work, we will instead select the value of s based on the best outcome in terms of performance and signature size.

[3] Again, for convenience, we choose $\mathbf{A}_0 = \mathbf{I}_m$, $\mathbf{B}_0 = \mathbf{I}_n$.

Remark 2. This optimization comes at the cost of an s-fold increase in public-key size. As shown for instance in [23], it would be possible to reduce this impact by using Merkle trees to a hash of the tree commitment of all the public keys. This, however, would add some significant overhead to the signature size, because it would be necessary to include the paths for all openings. Considering the sizes of the objects involved, such an optimization is not advantageous in our case.

Partially Seeding the Public Key. The previous optimization comes at significant cost to the public key, so we propose a new optimization that trades public key size for private key size. This optimization is inspired by the trade-off in the key generation of Rainbow [24] and UOV [11]. It has not been previously used in Fiat-Shamir signatures, but we expect it can be used successfully in any scheme coming from equivalence problems especially the ones using the previous optimization, such as [8,13,23]. With this optimization, 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 $\mathbf{G}_i$. In more detail, in order to generate the public $\mathbf{G}_i$ and the corresponding secret $(\mathbf{A}_i, \mathbf{B}_i)$ we perform the following:

- We perform a secret change of basis of $\mathbf{G}_0$ by multiplying it by a secret matrix $\mathbf{T} \in \mathrm{GL}_k(q)$ to obtain $\mathbf{G}'_0$. Assume the codewords from $\mathbf{G}'_0$ are $\mathbf{P}^0_1, \mathbf{P}^0_2, \ldots, \mathbf{P}^0_k$.
- For each $i \in \{1, \ldots, s-1\}$, we generate from a public seed a complete $m \times n$ codeword $\mathbf{P}^i_1$ and the top $m-1$ rows of codeword $\mathbf{P}^i_2$ (depending on the parameters m, n one can get slightly more rows when $m \neq n$).
- Find $\mathbf{A}_i$ and $\mathbf{B}_i$ from the linear relations:

$$\mathbf{P}^i_1 \mathbf{B}_i^{-1} = \mathbf{A}_i \mathbf{P}^0_1$$
$$\mathbf{P}^i_2 \mathbf{B}_i^{-1} = \mathbf{A}_i \mathbf{P}^0_2$$

  by fixing the first (top left) value of $\mathbf{A}_i$.
- Find $\mathbf{P}^i_j = \mathbf{A}_i \mathbf{P}^0_j \mathbf{B}_i$ for all $j \in \{3, \ldots, k\}$.
- Construct the public $\mathbf{G}_i$ from $\mathbf{P}^i_1, \mathbf{P}^i_2, \ldots, \mathbf{P}^i_k$.

The public key then is the public seed together with $\mathbf{P}^i_3, \ldots, \mathbf{P}^i_k$. For verification, the complete $\mathbf{G}_i$ are reconstructed using the seed.

Fixed-Weight Challenges. Another common optimization is the use of fixed-weight challenges. The idea is to generate the challenge string h with a fixed number of 1s and 0s, i.e. Hamming weight, rather than uniformly at random. This is because, when $h_i = 0$, the response (μ_i, ν_i) consists entirely of randomly-generated objects, and so one can just transmit the seed used for generating them. This creates a noticeable imbalance between the two types of responses, and hence it makes sense to minimize the number of 1 values. To this end, one can utilize a so-called *weight-restricted hash function*, that outputs values in $\mathbb{Z}^t_{2,w}$, by which we denote the set of vectors with elements in $\{0, 1\}$ of length t and weight w. In this way, although the length of the challenge strings increases,

Public Data

$q, m, n, k, \lambda, t, s, w \in \mathbb{N}$.
$\mathsf{hash} : \{0,1\}^* \to \{0,1\}^\lambda$.

I. Keygen()

1. $\mathbf{G}_0 \xleftarrow{\$} \mathbb{F}_q^{k \times mn}$ in standard form.
2. Set $\mathbf{A}_0 = \mathbf{I}_m$, $\mathbf{B}_0 = \mathbf{I}_n$.
3. $\mathbf{T} \xleftarrow{\$} \mathrm{GL}_k(q)$
4. Compute $\mathbf{G}'_0 = \mathbf{T}\mathbf{G}_0$
5. Parse the first two rows of $\mathbf{G}'_0$ into $\mathbf{P}_1^0, \mathbf{P}_2^0 \in \mathbb{F}_q^{m \times n}$
6. For all $j = 1, \ldots, s-1$:
   i. $\mathbf{P}_1^j, \mathbf{P}_2^j \xleftarrow{\$} \mathbb{F}_q^{m \times n}$
   ii. Find $\mathbf{A}_j$ and $\mathbf{B}_j$ from:
   $$\mathbf{P}_1^j \mathbf{B}_j^{-1} = \mathbf{A}_j \mathbf{P}_1^0$$
   $$\mathbf{P}_2^j \mathbf{B}_j^{-1} = \mathbf{A}_j \mathbf{P}_2^0$$
   iii. Compute $\mathbf{G}_j = \mathsf{SF}(\pi_{\mathbf{A}_j, \mathbf{B}_j}(\mathbf{G}'_0))$.
7. Set $\mathsf{sk} = (\mathbf{A}_1^{-1}, \mathbf{B}_1^{-1}, \ldots, \mathbf{A}_{s-1}^{-1}, \mathbf{B}_{s-1}^{-1})$.
8. Set $\mathsf{pk} = (\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_{s-1})$.

II. Sign(sk)

1. For all $i = 0, \ldots, t-1$:
   i. $\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i \xleftarrow{\$} \mathrm{GL}_m(q) \times \mathrm{GL}_n(q)$.
   ii. Compute $\tilde{\mathbf{G}}_i = \mathsf{SF}(\pi_{\tilde{\mathbf{A}}_i, \tilde{\mathbf{B}}_i}(\mathbf{G}_0))$.
2. Compute $h = \mathsf{hash}(\tilde{\mathbf{G}}_0, \ldots, \tilde{\mathbf{G}}_{t-1}, \mathsf{msg})$.
3. Expand h to $(h_0, \ldots, h_{t-1})$, $0 \le h_i < s$.
4. For all $i = 0, \ldots, t-1$:
   i. Set $(\mu_i, \nu_i) = (\tilde{\mathbf{A}}_i \mathbf{A}_{h_i}^{-1}, \mathbf{B}_{h_i}^{-1} \tilde{\mathbf{B}}_i)$.
5. Set $\sigma = (\mu_0, \ldots, \mu_{t-1}, \nu_0, \ldots, \nu_{t-1}, h)$.
6. Send σ to verifier.

III. Verify(pk, msg, σ)

1. Expand h to $(h_0, \ldots, h_{t-1})$, $0 \le h_i < s$.
2. For all $i = 0, \ldots, t-1$:
   i. Set $\hat{\mathbf{G}}_i = \mathsf{SF}(\pi_{\mu_i, \nu_i}(\mathbf{G}_{h_i}))$.
3. Compute $h' = \mathsf{hash}(\hat{\mathbf{G}}_0, \ldots, \hat{\mathbf{G}}_{t-1}, \mathsf{msg})$.
4. Accept if $h' = h$ or reject otherwise.

Fig. 3. The MEDS Protocol

the overall communication cost scales down proportionally to the value of w. In terms of security, this optimization only entails a small modification in the statement of the Forking Lemma, and it is enough to choose parameters such that $\log_2 \binom{t}{w} \ge \lambda$. Note that this optimization can easily be combined with the previous one, by mandating hash digests in $\mathbb{Z}_{s,w}^t$ and choosing parameters such that $\log_2 \left(\binom{t}{w}(s-1)^w\right) \ge \lambda$. In practice, this can be achieved with a hash function $\mathsf{hash} : \{0,1\}^* \to \{0,1\}^\lambda$, by expanding the output to a t-tuple $(h_0, \ldots, h_{t-1})$, $0 \le h_i < s$ of weight w.

Seed Tree. Finally, the signature size can be optimized again using a *seed tree*. This primitive allows to generate the many seeds used throughout the protocol in a recursive way, starting from a master seed mseed and building a binary tree, via repeated PRNG applications, having t seeds as leaves. When the required $t - w$ values need to be retrieved, it is then enough to reveal the appropriate sequence of nodes. This reduces the space required for the seeds from $\lambda(t-w)$ to λN_{seeds}, where N_{seeds} can be upper bounded by $2^{\lceil \log_2(w) \rceil} + w(\lceil \log_2(t) \rceil - \lceil \log_2(w) \rceil - 1)$, as shown in [29]. We refer the reader to Section 2.7 of [12] for more details. As suggested in [12], we are including a 256-bit salt to ward off multi-target collision attacks and the leaf address as identifier for domain separation in the inputs of the seed-tree hash functions.

To give a complete picture, we present the MEDS protocol in Fig. 3, in its final form, including all applicable variants. The various parameters control different

optimization: for instance s refers to the number of public keys used, whereas w refers to the fixed weight of the challenge hash string. Parameter choices will be thoroughly discussed in Sect. 7.1.

Public Key and Signature Size. With these various optimizations, we obtain the following public key and signature size for MEDS:

- MEDS public key size: $\lambda + (s-1)((k-2)(mn-k)+n)\lceil \log_2(q) \rceil$
- MEDS signature size:

$$\underbrace{\lambda}_{h} + \underbrace{w(m^2+n^2)\lceil \log_2(q) \rceil}_{\{\mu_i,\nu_i\}_{h_i=1}} + \underbrace{\lambda N_{\text{seeds}}}_{\{\mu_i,\nu_i\}_{h_i=0}} + \underbrace{2\lambda}_{salt}$$

6 Concrete Security Analysis

In this section, we will mostly use the Big O notation $\mathcal{O}$ to express the complexity of algorithms. Where we are not interested in the polynomial factor we will use $\mathcal{O}^*$. We note that despite the notation, the estimates are quite tight and provide a good basis for choosing parameters.

Recall that the goal of an adversary against MCE is to recover the matrices $\mathbf{A}$ and $\mathbf{B}$, given a description of the matrix codes $\mathcal{C}$ and $\mathcal{D}$. The most naïve attack would be to try every $\mathbf{A} \in \mathrm{GL}_m(q)$ and $\mathbf{B} \in \mathrm{GL}_n(q)$ until we find the correct isometry, amounting to a complexity of $\mathcal{O}(q^{n^2+m^2})$.

The naïve attack can be improved by noting that once one of the matrices $\mathbf{A}$ or $\mathbf{B}$ is known, the resulting problem becomes easy [22]. Hence, we only need to brute-force one of $\mathbf{A}$ or $\mathbf{B}$, so the complexity becomes $\mathcal{O}^*(q^{\min\{m^2,n^2\}})$.

In the rest of the section, we will see that there exist several non-trivial attacks that perform much better than this upper bound.

6.1 Birthday-Based Graph-Theoretical Algorithms for Solving MCE

Recent works [22,45] investigate the hardness of MCE by connecting it to other equivalence problems, namely, the Code Equivalence problem in the Hamming metric [22] and the Quadratic Maps Linear Equivalence problem (QMLE) [45]. The latter provides complexity analysis by viewing MCE as an instance of QMLE. We recap their results here. For better understanding, we include the definition of the related QMLE problem.

Problem 4. $\mathsf{QMLE}(k, N, \mathcal{F}, \mathcal{P})$:
Given: Two k-tuples of multivariate polynomials of degree 2

$$\mathcal{F} = (f_1, f_2, \ldots, f_k),\ \mathcal{P} = (p_1, p_2, \ldots, p_k) \in \mathbb{F}_q[x_1, \ldots, x_N]^k.$$

Goal: Find – if any – matrices $\mathbf{S} \in \mathrm{GL}_N(q), \mathbf{T} \in \mathrm{GL}_k(q)$ such that

$$\mathcal{P}(\mathbf{x}) = (\mathcal{F}(\mathbf{x}\mathbf{S}))\mathbf{T}.$$

Algorithm 1 Collision-search algorithm

```
1: function BuildList(F, P)
2:     L ← ∅
3:     repeat
4:         x ←$ F_q^(m+n)
5:         if P(F, x) then L ← L ∪ {x}
6:     until |L| = ℓ
7:     return L
8: function CollisionFind(F, P)
9:     L1 ← BuildList(F, P)
10:    L2 ← BuildList(P, P)
11:    for all (x, y) ∈ {L1 × L2} do
12:        φ ←inhQMLE(x, y)
13:        if φ ≠ ⊥ then
14:            return solution φ
15:    return ⊥
```

We denote by hQMLE, inhQMLE and BMLE the related problems when the polynomials are homogeneous of degree 2, inhomogeneous and bilinear, respectively. It was shown in [45] that, under the assumption that the two codes $\mathcal{C}$ and $\mathcal{D}$ have trivial automorphism groups (which is believed to be true with overwhelming probability for big enough parameters), $\mathsf{MCE}(k, n, m, \mathcal{C}, \mathcal{D})$ is equivalent to $\mathsf{hQMLE}(k, N, \mathcal{F}, \mathcal{P})$ where $N = m + n$. Concretely, an MCE instance with a solution $(\mathbf{A}, \mathbf{B})$ is transformed into an hQMLE instance with a solution $(\mathbf{S}, \mathbf{T})$ where $\mathbf{S} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^\top \end{bmatrix}$ and $\mathbf{T}$ corresponds to a change of basis of $\mathcal{D}$. Therefore it is possible to apply algorithms for solving hQMLE to MCE instances such as the graph-theoretic algorithm of Bouillaguet et al. [17]. The algorithm is basically a collision-search algorithm comprised of two steps, as given in Algorithm 1. In the first step we build two lists L_1 and L_2 of size ℓ of elements in $\mathbb{F}_q^{(m+n)}$ that satisfy a predefined distinguishing property $\mathbb{P}$ related to the given systems of polynomials $\mathcal{F}$ and $\mathcal{P}$ and that is preserved under isometry. In the second step, we try to find a collision between the two lists that will lead us to the solution. For the property $\mathbb{P}$, the authors of [17] propose:

$$\mathbb{P}(\mathcal{F}, \mathbf{x}) = \top \Leftrightarrow Dim(Ker(D_{\mathbf{x}}(\mathcal{F}))) = \kappa$$

for a suitably chosen κ, where $D_{\mathbf{x}}(\mathcal{F}) : \mathbf{y} \mapsto \mathcal{F}(\mathbf{x} + \mathbf{y}) - \mathcal{F}(\mathbf{x}) - \mathcal{F}(\mathbf{y})$ is the *differential* of $\mathcal{F}$ at a point $\mathbf{x}$. Clearly, the rank of the differential is preserved under isometry, so this is an appropriate choice of $\mathbb{P}$. Other instantiations are possible as well, as long as they are invariant under isometry, although their success depends on the distribution of elements that satisfy the property for varying κ.

Once a collision $(\mathbf{a}, \mathbf{b})$ is found, it can be used to derive an associated *inhomogeneous* QMLE instance inhQMLE $(k, (m + n), \mathcal{F}', \mathcal{P}')$ as $\mathcal{F}'(\mathbf{x}) = \mathcal{F}(\mathbf{x} + \mathbf{a})$, $\mathcal{P}'(\mathbf{x}) = \mathcal{P}(\mathbf{x} + \mathbf{b})$ on which we call an inhomogeneous solver. Since it can not be directly checked whether a pair is a collision, the solver needs to be called for each pair, similar to the guess and check approach in ISD algorithms [41].

The inhomogeneous instance can be solved much more efficiently than the homogeneous one. Heuristic evidence suggests that solving *random* instances of the inhQMLE problem using an algebraic approach takes $\mathcal{O}((m + n)^9)$ operations [27], however, the derived inhQMLE instances from the collision-search attack are not random enough. These specific instances have a solver with a

complexity of $\mathcal{O}(q^\kappa)$ [16]. As κ is typically chosen to be small, this approach is still efficient in practice. Following the analysis from [45], the concrete complexity of the algorithm for $k \leqslant 2(m+n)$ follows a birthday argument and is the maximum of the complexity of the two steps, i.e.:

$$\mathsf{max}(\sqrt{q^{(m+n)}/d} \cdot C_{\mathbb{P}}, dq^{(m+n)} \cdot C_{\mathsf{iQ}}), \tag{1}$$

with success probability of $\approx 63\%$. Here, $C_{\mathbb{P}}$ denotes the cost of checking whether an element satisfies the property $\mathbb{P}$, d is the proportion of elements satisfying $\mathbb{P}$ and C_{iQ} denotes the cost of a single query to $\mathsf{inhQMLE}$. Note that d can be calculated as $d = 1/\mathcal{O}(q^{\kappa^2+\kappa(k-(m+n))})$ and κ is chosen such that it minimizes Eq. (1). Asymptotically, the complexity is $\mathcal{O}^*(q^{\frac{2}{3}(m+n)})$ by balancing the steps [45]. The memory complexity is simply the size of the lists.

It is pointed out in [45] that when $k \geq 2(m+n)$, we can no longer assume that we have any elements satisfying $\mathbb{P}$, which forces us to consider *all* elements in the collision search giving a complexity of $\mathcal{O}(q^{m+n})$. In that case, we can consider choosing arbitrarily one element $\mathbf{x}$ and checking for a collision with all other elements $\mathbf{y} \in \mathbb{F}_q^{m+n}$. Note that this approach was also proposed in [17], and can be applied to any parameter set, thus giving an upper-bound on the complexity of a classical collision-search algorithm.

For a quantum version of Algorithm 1, both BuildList and CollisionFind can be seen as searches of unstructured databases of a certain size, hence Grover's algorithm applies to both: we can build the list L using only $\sqrt{\ell \cdot d^{-1}}$ searches, and we can find a collision using only $\sqrt{|L_1 \times L_2|}$ queries to the solver. This requires both $\mathbb{P}$ and $\mathsf{inhQMLE}$ to be performed in superposition. The balance between both sides remains the same. In total, the complexity of the quantum version becomes $\mathcal{O}^*(q^{\frac{1}{3}(m+n)})$.

Collision-Search Algorithm Using Non-trivial Roots. When viewing an MCE instance as an hQMLE instance, it is possible to use certain bilinear properties to improve Algorithm 1. When $n = m$, such instances have approximately q^{2n-k-1} non-trivial roots, which can be used to improve a subroutine of Algorithm 1, and to make it deterministic instead of probabilistic [45]. In practice, such non-trivial roots exist **i)** almost always when $k < 2n$, **ii)** with probability $1/q$ for $k = 2n$, **iii)** with probability $1/q^{k+1-2n}$ for $k > 2n$. The complexity of this approach is $\mathcal{O}^*(q^n)$, if such non-trivial roots exist. This complexity is proven under the assumption that the complexity of the inhomogenous QMLE solver is no greater than $\mathcal{O}(q^n)$, which holds trivially when $k \geq n$ [45], and heuristically when $k < n$. Finding the non-trivial roots can also be done using a bilinear XL algorithm [40]. We do not consider this approach in our analysis, as it is only interesting for a subset of parameters where the systems are (over)determined, i.e. when k is close to $m+n$.

6.2 Algebraic Attacks

Direct Modelling. Recently, in [45], it was shown that MCE is equivalent to BMLE. One of the natural attack avenues is thus to model the problem as an algebraic system of polynomial equations over a finite field. This approach was taken in [27], where the general Isomorphism of Polynomials (IP) problem was investigated. Here, we focus specifically on BMLE and perform a detailed complexity analysis.

First, fix arbitrary bases $(\mathbf{C}^{(1)}, \ldots, \mathbf{C}^{(k)})$ and $(\mathbf{D}^{(1)}, \ldots, \mathbf{D}^{(k)})$ of the codes $\mathcal{C}$ and $\mathcal{D}$ respectively. In terms of the bases, the MCE problem can be rephrased as finding $\mathbf{A} \in \mathrm{GL}_m(q), \mathbf{B} \in \mathrm{GL}_n(q)$ and $\mathbf{T} = (t_{ij}) \in \mathrm{GL}_k(q)$ such that:

$$\sum_{1 \leqslant s \leqslant k} t_{rs} \mathbf{D}^{(s)} = \mathbf{A}\mathbf{C}^{(r)}\mathbf{B}, \quad \forall r, 1 \leqslant r \leqslant k \tag{2}$$

The system (2) consists of knm equations in the $m^2+n^2+k^2$ unknown coefficients of the matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{T}$. The quadratic terms of the equations are always of the form $\gamma a_{ij} b_{i'j'}$ for some coefficients a_{ij} and $b_{i'j'}$ of $\mathbf{A}$ and $\mathbf{B}$ respectively which means the system (2) is bilinear. Note that the coefficients of $\mathbf{T}$ appear only linearly. As previously, we can guess the m^2 variables from $\mathbf{A}$, which will lead us to a linear system that can be easily solved. However, we can do better by exploiting the structure of the equations.

For ease of readability of the rest of the paragraph denote by $\mathbf{M}_{i\_}$ and $\mathbf{M}_{\_i}$ the i-th row and i-th column of a matrix $\mathbf{M}$. Note that, in (2), for $i \neq j$, the unknown coefficients from two rows $\mathbf{A}_{i\_}$ and $\mathbf{A}_{j\_}$ don't appear in the same equation. Symmetrically, the same holds for $\mathbf{B}_{\_i}$ and $\mathbf{B}_{\_j}$, but we will make use of it for the matrix $\mathbf{A}$. Thus, we can consider only part of the system, and control the number of variables from $\mathbf{A}$. The goal is to reduce the number of variables that we need to guess before obtaining an overdetermined linear system, and we want to do this in an optimal way. Consider the first α rows from $\mathbf{A}$. Extracting the equations that correspond to these rows in (2) leads us to the system:

$$\sum_{1 \leqslant s \leqslant k} t_{rs} \mathbf{D}^{(s)}_{i\_} = \mathbf{A}_{i\_}\mathbf{C}^{(r)}\mathbf{B}, \quad \forall r, i, \ 1 \leqslant r \leqslant k, \ 1 \leqslant i \leqslant \alpha. \tag{3}$$

Guessing the αm coefficients from $\mathbf{A}_{i\_}$ leads to a linear system of αkn equations in $n^2 + k^2$ variables. Choosing $\alpha = \lceil \frac{n^2+k^2}{kn} \rceil$, the complexity of the approach becomes $\mathcal{O}(q^{m\lceil \frac{n^2+k^2}{kn} \rceil}(n^2 + k^2)^3)$. For the usual choice of $m = n = k$, this reduces to at least $\alpha = 2$ and a complexity of $\mathcal{O}(q^{2n}n^6)$.

Note that, one can directly solve the bilinear system (3) using for example XL [20] and the analysis for bilinear systems from [40] (similar results can be obtained from [25]). We have verified, however, that due to the large number of variables compared to the available equations, the complexity greatly surpasses the one of the simple linearization attack presented above.

Improved Modelling. In order to improve upon this baseline algebraic attack, we will model the problem differently and completely avoid the t_{rs} variables. This modelling is in the spirit of the minors modellings of MinRank as in [7,26].

As previously, let $\mathbf{G}$ and $\mathbf{G}'$ be the $k \times mn$ generator matrices of the equivalent codes $\mathcal{C}$ and $\mathcal{D}$ respectively. Then from Lemma 1, $\tilde{\mathbf{G}} = \mathbf{G}(\mathbf{A}^\top \otimes \mathbf{B})$ is a generator matrix of $\mathcal{D}$ for some invertible matrices $\mathbf{A}$ and $\mathbf{B}$. We will take the coefficients of $\mathbf{A}$ and $\mathbf{B}$ to be our unknowns. A crucial observation for this attack is that each row $\tilde{\mathbf{G}}_{i\_}$ of $\tilde{\mathbf{G}}$ is in the span of the rows of $\mathbf{G}'$, since $\mathbf{G}'$ and $\tilde{\mathbf{G}}$ define the same code. This means that adding $\tilde{\mathbf{G}}_{i\_}$ to $\mathbf{G}'$ does not change the code, i.e.,

$$ {}^{(i)}\mathbf{G}' = \begin{pmatrix} \mathbf{G}' \\ \tilde{\mathbf{G}}_{i\_} \end{pmatrix} $$

is not of full rank. From here, all maximal minors $| \left({}^{(i)}\mathbf{G}'_{\_j_1} \; {}^{(i)}\mathbf{G}'_{\_j_2} \ldots {}^{(i)}\mathbf{G}'_{\_j_{k+1}} \right) |$ of ${}^{(i)}\mathbf{G}'$, for every $\{j_1, j_2, \ldots, j_{k+1}\} \subset \{1, 2, \ldots, mn\}$, are zero.

Now, as in a minors modeling of MinRank, we can form equations in the unknown coefficients of $\mathbf{A}$ and $\mathbf{B}$ by equating all maximal minors to zero, which amounts to a total of $\binom{mn}{k+1}$ equations. Since the unknown coefficients of $\mathbf{A}$ and $\mathbf{B}$ appear only in the last row of the minors, and only bilinearly, the whole system is also bilinear. Thus we have reduced the problem to solving the bilinear system

$$ \left\{ | \left({}^{(i)}\mathbf{G}'_{\_j_1} \; {}^{(i)}\mathbf{G}'_{\_j_2} \ldots {}^{(i)}\mathbf{G}'_{\_j_{k+1}} \right) | = 0, \begin{array}{l} \text{for all } i \in \{1, 2, \ldots, k\} \text{ and all} \\ \{j_1, j_2, \ldots, j_{k+1}\} \subset \{1, 2, \ldots, mn\} \end{array} \right. \tag{4} $$

in the $m^2 + n^2$ unknown coefficients of $\mathbf{A}$ and $\mathbf{B}$.

At first sight, (4) seems to have more than enough equations to fully linearize the system. However, the majority of these equations are linearly dependent. In fact, there are only $(mn - k)k$ linearly independent equations. To see this, fix some i and consider a minor $| \left({}^{(i)}\mathbf{G}'_{\_j_1} \; {}^{(i)}\mathbf{G}'_{\_j_2} \ldots {}^{(i)}\mathbf{G}'_{\_j_{k+1}} \right) |$ of ${}^{(i)}\mathbf{G}'$. Since all rows except the first don't contain any variables, the equation

$$ | \left({}^{(i)}\mathbf{G}'_{\_j_1} \; {}^{(i)}\mathbf{G}'_{\_j_2} \ldots {}^{(i)}\mathbf{G}'_{\_j_{k+1}} \right) | = 0 $$

basically defines the linear dependence between the columns ${}^{(i)}\mathbf{G}'_{\_j_1}, \ldots {}^{(i)}\mathbf{G}'_{\_j_{k+1}}$. But the rank of the matrix is k, so all columns can be expressed through some set of k independent columns. Thus, in total, for a fixed i we have $mn - k$ independent equations and in total $(mn - k)k$ equations for all i.

Alternatively, we can obtain the same amount of equations from $\tilde{\mathbf{G}}$ and the generator matrix $\mathbf{G}'^{\perp}$ of the dual code of $\mathcal{D}$. Since $\tilde{\mathbf{G}}$ should also be a generator matrix of $\mathcal{D}$, we construct the system:

$$ \mathbf{G}'^{\perp} \cdot \tilde{\mathbf{G}}^\top = \mathbf{0}, $$

which is again a system of $(mn - k)k$ bilinear equations in $n^2 + m^2$ variables.

The complexity of solving the obtained system using either of the modellings strongly depends on the dimension of the code – it is the smallest for $k = mn/2$,

and grows as k reduces (dually, as k grows towards mn). In Sect. 7 we give the concrete complexity estimate for solving the system for the chosen parameters using bilinear XL and the analysis from [40].

The attack does not seem to benefit a lot from being run on a quantum computer. Since the costly part comes from solving a huge linear system for which there are no useful quantum algorithms available, the only way is to 'Groverize' an enumeration part of the algorithm. One could enumerate over one set of the variables, either of $\mathbf{A}$ or $\mathbf{B}$, typically the smaller one, and solve a biliner system of less variables. Grover's algorithm could then speed up quadratically this enumeration. However, since in the classical case the best approach is to not use enumeration, this approach only makes sense for quite small values of the field size i.e. only when $q < 4$. In this parameter regime, however, combinatorial attacks perform significantly better, so this approach becomes irrelevant.

6.3 Leon-Like Algorithm Adapted to the Rank Metric

Leon [34] proposed an algorithm against the code equivalence problem in the Hamming metric that relies on the basic property that isometries preserve the weight of the codewords and that the weight distribution of two equivalent codes is the same. Thus, finding the set of codewords of smallest weight in both codes reveals enough information to find a permutation that maps one set to the other, which with high probability is the unknown isometry between the codes. This algorithm is quite unbalanced and heavy on the 'codewords finding' side, since it requires finding all codewords of minimal weight. Beullens [10] proposed to relax the procedure and instead perform a collision based algorithm, much in the spirit of Algorithm 1: Build two lists of elements of the codes of particular weight (the distinguishing property from [10] actually also includes the multiset of entries of a codeword) and find a collision between them. As in Leon's algorithm and Algorithm 1, the 'collision finding' part employs an efficient subroutine for reconstructing the isometry.

The approach from the Hamming metric can be translated to matrix codes and can be used to solve MCE, but some adjustments are necessary. First of all note that finding codewords of a given rank r is equivalent to an instance of MinRank [19,26] for k matrices of size $m \times n$ over $\mathbb{F}_q$. Depending on the parameters, we have noticed that the Kipnis-Shamir modelling [32] and Bardet's modelling [7] perform the best, so we use both in our complexity estimates.

For the collision part, notice that given two codewords $\mathbf{C}_1$ from $\mathcal{C}$ and $\mathbf{D}_1$ from $\mathcal{D}$, it is not possible to determine the isometry $(\mathbf{A}, \mathbf{B})$, as there are many isometries possible between single codewords. Thus, there is no efficient way of checking that these codewords collide nor finding the correct isometry. On the other hand, a pair of codewords is typically enough. For the pairs $(\mathbf{C}_1, \mathbf{C}_2)$ and $(\mathbf{D}_1, \mathbf{D}_2)$ we can form the system of $2mn$ linear equations

$$\begin{cases} \mathbf{A}^{-1}\mathbf{D}_1 = \mathbf{C}_1\mathbf{B} \\ \mathbf{A}^{-1}\mathbf{D}_2 = \mathbf{C}_2\mathbf{B} \end{cases} \tag{5}$$

in the $m^2 + n^2$ unknown coefficients of $\mathbf{A}$ and $\mathbf{B}$. When $m = n$, which is a typical choice, the system is expected to be overdetermined, and thus solved in $\mathcal{O}(n^6)$. In practice, and since $\mathbf{C}_1$, $\mathbf{C}_2$, $\mathbf{D}_1$ and $\mathbf{D}_2$ are low-rank codewords, there are fewer than $2n^2$ linearly independent equations, so instead of a unique solution, we can obtain a basis of the solution space. However, the dimension of the solution space is small enough so that coupling this technique with one of the algebraic modelings in Sect. 6.2 results in a system that can be solved through direct linearization. It is then easy to check whether the obtained isometry maps $\mathcal{C}$ to $\mathcal{D}$. We will thus assume, as a lower bound, that we find collisions between pairs of codewords.

Now, let $C(r)$ denote the number of codewords of rank r in a k-dimensional $m \times n$ matrix code. Then, using a birthday argument, two lists of size $\sqrt{2C(r)}$ of rank r codewords of $\mathcal{C}$ and $\mathcal{D}$ are enough to find two collisions. To detect the two collisions, we need to generate and solve systems as in Eq. (5) for all possible pairs of elements from the respective lists, so $\binom{\sqrt{2C(r)}}{2}^2$ systems in total. Since $C(r) \approx q^{r(n+m-r)-nm+k}$, the total complexity amounts to

$$\mathcal{O}(q^{2(r(n+m-r)-nm+k)}(m^2+n^2)^\omega).$$

Note that a deterministic variant of this approach has the same asymptotic complexity. Choosing two rank r codewords of $\mathcal{C}$ and checking them for a 2-collision against all pairs of rank r codewords of $\mathcal{D}$ requires solving $\binom{C(r)}{2}$ systems.

Finally, we choose r so that both parts – the MinRank and the collision part are as close to a balance as possible. Section 7 discuses further the complexity of this approach for the chosen parameters of our scheme.

When considering the quantum version of the algorithm, we apply the same reasoning as in the case of the collision based Algorithm 1, and obtain quadratic speedup in the collision part. Because hybridization is also possible for the MinRank part, it can also benefit from using Grover, especially for larger fields.

7 Implementation and Evaluation

In this section we give an assessment of the performance of MEDS. We provide concrete parameter choices for MEDS and a first preliminary evaluation of its performance based on a C reference implementation as well as a comparison to related signature schemes. The source code of our implementation is available at https://github.com/MEDSpqc/meds.

For our reference implementation, we simply implemented all finite field arithmetic in $\mathbb{F}_q$ using integer arithmetic modulo q, where q is a prime. We implemented all matrix multiplication, generating random invertible matrices, and computing the systematic form of a matrix in constant time such that their runtime does not depend on secret input.

We are using two different approaches for generating an invertible matrix M: We either generate a random matrix and check if it is invertible by explicitly computing its inverse or we construct an invertible matrix following the approach

Table 1. Cost of the investigated attacks in log scale, and 'SIG' for 'signature size in bytes. Preferred choice in bold.

$\lceil \log_2 q \rceil$	$n = k$	Birthday	Algebraic	Leon	SIG
9	16	235.29	181.55	131.20	13 296
9	17	249.04	194.55	149.65	16 237
10	15	244.62	174.75	130.50	12 428
11	14	250.79	160.24	131.21	12 519
12	14	272.40	160.24	141.17	13 548
13	**13**	**274.10**	**146.76**	**130.41**	**11 586**
14	13	294.10	146.76	134.41	13 632
20	12	383.75	138.46	135.40	16 320

of [47] based on the approach by [43] by generating a random lower-left triangular matrix $\mathbf{L}$ with the diagonal all 1 and an upper-right triangular matrix $\mathbf{U}$ with the diagonal all $\neq 0$ and computing $\mathbf{M}$ as $\mathbf{M} = \mathbf{LU}$ directly. This, however, covers only a subset of $((q-1)/q)^n$ matrices of all invertibe matrices in $\mathbb{F}_q^{n \times n}$. We are using the first approach for key generation, since here we need not only invertible matrices but also their inverses anyways, and the second approach for signing where the inverses of invertible matrices are not explicitly required.

7.1 Parameter Choice and Evaluation

A summary of the cost of the three different attacks described in Sect. 6 is given in Table 1. First, we decide to set $n = k$, as this seems to be the Goldilocks zone for our scheme. For k larger, the algebraic attack becomes significantly better, and the same is true for Leon's attack when k is smaller. Then, for finite fields of different sizes, we find the smallest value of n that achieves the required security level of 128 bits. We see that Leon's algorithm performs the best in most cases, although the algebraic approach is almost as good. Finally, to determine the optimal value for q, we choose the optimization parameters (s, t, and w) such that the sizes of the public key and the signature are comparable, and we report the signature size in the last column of Table 1. We conclude that the sweet spot for 128-bit security is given for the 13-bit prime $q = 8191$ and $n = k = 13$.

Remark 3. Given these parameters, we heuristically assume that the automorphism group of the codes is trivial with overwhelming probability. It is computationally infeasible to compute the automorphism group of codes of this size; however, data on smaller-sized codes shows that the probability of a random code having a trivial automorphism group grows rapidly as q, n, and m increase.

In this setting, we can vary s, t, and w for different trade-offs of public key and signature sizes as well as performance. We also checked the impact of q if we aim for small public keys or small signatures (instead if balancing these two as in Table 1). In such cases, both 11-bit and 13-bit primes for q seem to perform similarly well. Hence, we stick to the 13-bit prime $q = 8191$ in our discussion.

Table 2. Parameters for MEDS, for $\lambda = 128$ bits of classical security. 'ST' for seed tree. 'PK' for 'public key size' and 'SIG' for 'signature size in bytes, 'FS' for 'Fiat-Shamir' probability logarithmic to base 2.

Parameter Set	q	n	m	k	s	t	w	ST	PK	SIG	FS
MEDS-2826-st	8191	13	13	13	2	256	30	✓	2826	18020	−129.739
MEDS-8445-st-f	8191	13	13	13	4	160	23	✓	8445	13946	−128.009
MEDS-8445-st	8191	13	13	13	4	464	17	✓	8445	10726	−128.764
MEDS-8445-st-s	8191	13	13	13	4	1760	13	✓	8445	8702	−128.162
MEDS-11255-st	8191	13	13	13	5	224	19	✓	11255	11618	−128.451
MEDS-11255	8191	13	13	13	5	224	19	–	11255	13778	−128.451
MEDS-42161-st	8191	13	13	13	16	128	16	✓	42161	9616	−128.849
MEDS-356839-st	8191	13	13	13	128	80	12	✓	356839	7288	−129.64
MEDS-716471-st	8191	13	13	13	256	64	11	✓	716471	6530	−127.374

Table 3. Performance of MEDS in time (ms) and mega cycles (mcyc.) at 1900 MHz on an AMD Ryzen 7 PRO 5850U CPU following the SUPERCOP setup (https://bench.cr.yp.to/supercop.html) computed as median of 16 randomly seeded runs each.

Parameter Set	Key Generation		Signing		Verification	
	(ms)	(mcyc.)	(ms)	(mcyc.)	(ms)	(mcyc.)
MEDS-2826-st	71.128110	135.143409	102.787710	195.296649	98.00434	186.208246
MEDS-8445-st-f	211.447740	401.750706	63.206200	120.09178	60.13987	114.265753
MEDS-8445-st	211.354280	401.573132	185.680270	352.792513	178.42456	339.006664
MEDS-8445-st-s	211.766000	402.3554	697.002740	1324.305206	673.18607	1279.053533
MEDS-11255-st	258.177820	490.537858	88.123950	167.435505	84.46502	160.483538
MEDS-11255	258.988880	492.078872	88.191290	167.563451	84.50302	160.555738
MEDS-42161-st	969.972890	1842.948491	50.544150	96.033885	48.4196	91.99724
MEDS-356839-st	8200.832680	15581.582092	31.630390	60.097741	32.37874	61.519606
MEDS-716471-st	18003.067490	34205.828231	25.568960	48.581024	28.93696	54.980224

Table 2 provides an overview of 128-bit security parameters for MEDS, highlighting different performance and key/signature size trade-offs. The best attack for all parameter set based on $q = 8191$, $n = 13$, and $k = 13$ is the Leon-like attack as shown in Table 1 with an expected cost of slightly over 2^{130} operations. The best quantum attack is obtained by Groverizing Leon's algorithm and has a cost of around 2^{88} operations. We select s, t, and w such that the probability of an attack on the Fiat-Shamir construction is around 2^{-128}. To improve the efficiency of vectorized implementations using SIMD instructions in the future, we select t as multiple of 16. In general, we are using all optimizations discussed in Sect. 5. However, we provide one parameter set without using the seed tree (without '-st' in the name of the parameter set).

Table 3 shows the resulting performance of these parameter sets from our constant-time C reference implementation on an AMD Ryzen 7 PRO 5850U CPU. The C reference implementation follows the implementation discussion

Table 4. Performance comparison to other relevant schemes (mcyc. rounded to three significant figures). Data marked with '(scop)' is from the SUPERCOP website. For SPHINCS+ we list results for the 'simple' variant.

Scheme	pk size (byte)	sig size (byte)	key gen (mcyc.)	sign (mcyc.)	verify (mcyc.)
ed25519 (scop)	32	64	0.048442	0.051300	0.182148
[35] dilithium2 (scop)	1312	2420	0.151339	0.363393	0.162999
[42] falcon512dyn (scop)	897	666	19.520464	0.880309	0.085587
[31] sphincsf128shake256 (scop)	32	16976	6.856442	220.279833	9.905358
[31] sphincss128shake256 (scop)	32	8080	217.958286	3502.227717	4.036804
[11] UOV ov-Ip	278432	128	2.903434	0.105324	0.090336
[8] LESS-I	8748	12728	—	—	—
[6] Wavelet	3236327	930	7403.069461	1644.281062	1.087538
[2] SDitH Var3f	144	12115	—	4.03000	3.0380
[2] SDitH Var3sss	144	5689	—	994.0460	969.2770
MEDS-8445-st-f	8445	13914	401.75	120.09	114.27
MEDS-11255-st	11255	11586	490.54	168.44	160.48
MEDS-42161-st	42161	9584	1842.95	96.03	92.00
MEDS-716471-st	716471	6498	34205.83	48.583	54.98

above but does not apply any further algorithmic or platform-specific optimizations. We expect that optimized and vectorized implementations can significantly increase the performance.

The parameter set MEDS-2826-st with $s = 2$ provides the smallest public key with about 2.8 kB and a signature of about 18 kB. MEDS-8445-st increases the public key size with $s = 4$ to slightly over 8 kB while reducing the signature size to about 10.5 kB. MEDS-8445-st-f is a 'fast' variant of this parameter set with a smaller $t = 160$ but a larger $w = 23$, resulting in a larger signature size of about 14 kB. MEDS-8445-st-s is 'small' and goes the opposite direction, providing a smaller signature size of about 8.5 kB due to a smaller $w = 13$ at a larger computational cost due to $t = 1760$. These three sibling parameter sets illustrate the impact of t and w on performance and signature size.

MEDS-11255-st provides balanced public key and signature sizes, with both around 11 kB, and a small sum of signature and public key size at moderate computational cost for signing and verification due to $t = 224$. Removing the seed tree optimization comes with an increase in signature size of about 2 kB, which illustrates the impact of the seed tree.

Finally, sets MEDS-42161-st, MEDS-356839-st, and MEDS-716471-st push the public key size to an extreme at the expense of key generation time in the pursue of reducing signature size and computational cost for signing and verification. However, we expect that at least the key generation time can significantly be improved by optimizing the computation of solving the medium-size sparse linear system used for partially seeding the public key.

7.2 Comparison to Related Signature Schemes

Table 4 shows a comparison of public key and signature sizes as well as computational performance of our new MEDS scheme with some established schemes and related recent proposals. While the comparison of public key and signature sizes is accurate, the comparison of the performance needs to be taken with a large grain of salt: While we provide numbers in the same performance metric (mega cycles – mcyc.), a direct comparison is still quite hard since not all schemes have received the same degree of optimization and since not all performance data has been obtained on the same CPU architecture.

The performance data from the 'classical' scheme ed25519 as well as from the NIST PQC schemes CRYSTALS-Dilithium [35], Falcon [42], and SPHNICS+ [31] has been obtained from the SUPERCOP website[4]. We selected the performance data from the AMD64 Zen CPU, which is an AMD Ryzen 7 1700 from 2017, i.e., the same microarchitecture (but a different CPU) as we used for our measurements of MEDS. We are reporting median cycles directly from the website.

For UOV [11], LESS [8] and Wavelet [6] we list the performance data as reported in the respective papers unless such data was unavailable. In the case of SDitH [2], only reports of performance data in milliseconds on a 3.1 GHz Intel Core i9-9990K are available. We computed the corresponding number of cycles from this to enable a rough comparison to the other schemes, but note that this data is therefore not entirely accurate.

Table 4 shows that, although code-based schemes do not compete well with pre-quantum or lattice-based PQC schemes, MEDS fills a gap that was not previously available for multivariate or code-based schemes, with a relatively small combined size of public key and signature. Furthermore, its versatility in parameter selection allows for great flexibility for specific applications. In terms of performance, the current implementation of MEDS is still unoptimized. We expect speed-ups of at least one order of magnitude from SIMD parallelization on AVX256 and AVX512 CPUs, since both the data-independent loop of the Fiat-Shamir construction and the matrix arithmetic lend themselves to efficient parallelization. Providing optimized implementations of MEDS for modern SIMD architectures as well as embedded systems is an open task for future work.

References

1. Aguilar Melchor, C., et al.: HQC. NIST PQC Submission (2020)
2. Aguilar-Melchor, C., Gama, N., Howe, J., Hülsing, A., Joseph, D., Yue, D.: The return of the SDitH. Cryptology ePrint Archive, Paper 2022/1645 (2022, to appear at Eurocrypt 2023)
3. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 411–439. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_14

[4] https://bench.cr.yp.to/results-sign.html – amd64; Zen (800f11); 2017 AMD Ryzen 7 1700; 8 × 3000 MHz; rumba7, supercop-20220506.

4. Albrecht, M.R., et al.: Classic McEliece. NIST PQC Submission (2020)
5. Aragon, N., et al.: BIKE. NIST PQC Submission (2020)
6. Banegas, G., Debris-Alazard, T., Nedeljković, M., Smith, B.: Wavelet: code-based postquantum signatures with fast verification on microcontrollers. Cryptology ePrint Archive, Paper 2021/1432 (2021)
7. Bardet, M., et al.: Improvements of algebraic attacks for solving the rank decoding and MinRank problems. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 507–536. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_17
8. Barenghi, A., Biasse, J.-F., Persichetti, E., Santini, P.: LESS-FM: fine-tuning signatures from the code equivalence problem. In: Cheon, J.H., Tillich, J.-P. (eds.) PQCrypto 2021 2021. LNCS, vol. 12841, pp. 23–43. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81293-5_2
9. Barenghi, A., Biasse, J.-F., Ngo, T., Persichetti, E., Santini, P.: Advanced signature functionalities from the code equivalence problem. Int. J. Comput. Math. Comput. Syst. Theory **7**(2), 112–128 (2022)
10. Beullens, W.: Not enough LESS: an improved algorithm for solving code equivalence problems over $\mathbb{F}_q$. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 387–403. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81652-0_15
11. Beullens, W., et al.: Oil and vinegar: modern parameters and implementations. Cryptology ePrint Archive, Paper 2023/059 (2023)
12. Beullens, W., Katsumata, S., Pintore, F.: Calamari and Falafl: logarithmic (linkable) ring signatures from isogenies and lattices. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 464–492. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_16
13. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 227–247. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_9
14. Biasse, J.-F., Micheli, G., Persichetti, E., Santini, P.: LESS is more: code-based signatures without syndromes. In: Nitaj, A., Youssef, A. (eds.) AFRICACRYPT 2020. LNCS, vol. 12174, pp. 45–65. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51938-4_3
15. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 493–522. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_17
16. Bouillaguet, C.: Algorithms for some hard problems and cryptographic attacks against specific cryptographic primitives. Ph.D. thesis, Université Paris Diderot (2011)
17. Bouillaguet, C., Fouque, P.-A., Véber, A.: Graph-theoretic algorithms for the "isomorphism of polynomials" problem. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 211–227. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_13
18. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15
19. Courtois, N.T.: Efficient zero-knowledge authentication based on a linear algebra problem MinRank. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 402–421. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_24

20. Courtois, N., Klimov, A., Patarin, J., Shamir, A.: Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 392–407. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_27
21. Couveignes, J.-M.: Hard homogeneous spaces. Cryptology ePrint Archive, Paper 2006/291 (2006)
22. Couvreur, A., Debris-Alazard, T., Gaborit, P.: On the hardness of code equivalence problems in rank metric. CoRR, abs/2011.04611 (2020)
23. De Feo, L., Galbraith, S.D.: SeaSign: compact isogeny signatures from class group actions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 759–789. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_26
24. Ding, J., et al.: Rainbow. Technical report, National Institute of Standards and Technology (2020)
25. Faugère, J.-C., Din, M.S.E., Spaenlehauer, P.-J.: Gröbner bases of bihomogeneous ideals generated by polynomials of bidegree (1, 1): algorithms and complexity. J. Symb. Comput. **46**(4), 406–437 (2011)
26. Faugère, J.-C., Levy-dit-Vehel, F., Perret, L.: Cryptanalysis of MinRank. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 280–296. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_16
27. Faugère, J.-C., Perret, L.: Polynomial equivalence problems: algorithmic and theoretical aspects. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 30–47. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_3
28. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
29. Gueron, S., Persichetti, E., Santini, P.: Designing a practical code-based signature scheme from zero-knowledge proofs with trusted setup. Cryptography **6**(1), 5 (2022)
30. Haviv, I., Regev, O.: On the lattice isomorphism problem. In: Chekuri, C. (ed.) SODA 2014, pp. 391–404. ACM SIAM (2014)
31. Hulsing, A., et al.: SPHINCS+. NIST PQC Submission (2020)
32. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE public key cryptosystem by relinearization. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 19–30. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_2
33. Kuperberg, G.: Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In: Severini, S., Brandão, F.G.S.L. (eds.) TQC 2013. LIPIcs, vol. 22, pp. 20–34. Schloss Dagstuhl (2013)
34. Leon, J.S.: Computing automorphism groups of error-correcting codes. IEEE Trans. Inf. Theory **28**(3), 496–510 (1982)
35. Lyubashevsky, V., et al.: CRYSTALS. NIST PQC Submission (2020)
36. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. DSN PR 42-44, California Institute of Technology (1978)
37. Nguyen, P., Wolf, C.: International workshop on post-quantum cryptography (2006)
38. NIST. Post-Quantum Cryptography Standardization (2017). https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
39. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_4

40. Perlner, R., Smith-Tone, D.: Rainbow band separation is better than we thought. Cryptology ePrint Archive, Paper 2020/702 (2020)
41. Prange, E.: The use of information sets in decoding cyclic codes. IRE Trans. Inf. Theory **8**(5), 5–9 (1962)
42. Prest, T., et al.: FALCON. NIST PQC Submission (2020)
43. Randall, D.: Efficient Generation of Random Nonsingular Matrices. Technical Report UCB/CSD-91-658, EECS Department, UC Berkeley (1991)
44. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) Theory of Computing, pp. 84–93. ACM (2005)
45. Reijnders, K., Samardjiska, S., Trimoska, M.: Hardness estimates of the code equivalence problem in the rank metric. Cryptology ePrint Archive, Paper 2022/276 (2022)
46. Rostovtsev, A., Stolbunov, A.: Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Paper 2006/145 (2006)
47. Tang, G., Duong, D.H., Joux, A., Plantard, T., Qiao, Y., Susilo, W.: Practical post-quantum signature schemes from isomorphism problems of trilinear forms. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022. LNCS, vol. 13277, pp. 582–612. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-07082-2_21