MAYO: Optimized Implementation with Revised Parameters for ARMv7-M

Arianna Gringiani[1], Alessio Meneghetti[1], Edoardo Signorini[3] and Ruggero Susella[2]

[1] University of Trento, Trento, Italy, a.gringiani@libero.it, alessio.meneghetti@unitn.it
[2] STMicroelectronics, Agrate Brianza, Italy, ruggero.susella@st.com
[3] Telsy, Torino, Italy, edoardo.signorini@telsy.it

Abstract. We present an optimized constant-time implementation of the MAYO signature scheme on ARMv7-M. MAYO is a novel multivariate proposal based on the trapdoor function of the Unbalanced Oil and Vinegar scheme. Our implementation builds on existing techniques for UOV-based schemes and introduces a new approach for evaluating the polar forms of quadratic maps. We modify MAYO's original parameters to achieve greater benefits from the proposed optimizations, resulting in slightly larger keys and shorter signatures for the same level of security. We evaluate the optimized implementation with the new parameters on the STM32H753ZIT6 microcontroller and measure its performance for the signing and verification procedures. At NIST security level I, signing requires approximately 43M cycles, and verification requires approximately 6M cycles. Both are 2.6 times faster than the results obtained from the original parameters.

Keywords: MAYO · post-quantum cryptography · multivariate quadratic cryptography · ARMv7-M

1 Introduction

Public key cryptography schemes, particularly key exchange and digital signatures, are a key building block in the realization of secure communication protocols. While current schemes have been deemed unbreakable in classical computational models, the rapid growth of quantum computers and the potential use of Shor's quantum algorithm [Sho94] threaten their security. In response, the American National Institute of Standards and Technology (NIST) initiated a standardization project in 2016 to establish new quantum-resistant public key cryptography standards. Following three selection rounds, in 2022 NIST selected four algorithms for standardization [AAC+22]: a key exchange algorithm (CRYSTALS-KYBER [ABD+21]) and three digital signature algorithms (CRYSTALS-DILITHIUM [BDK+21], Falcon [FHK+20] and SPHINCS+ [ABB+22]). Lattice-based schemes emerged as the most versatile, with three of the four winners falling under this category. NIST also expressed a desire to diversify the computational assumptions underlying these new standards, particularly for digital signatures, and issued a new call starting in 2023. [NIS23].

Multivariate quadratic-based constructions offer an effective alternative, producing highly efficient signature schemes with compact signatures, albeit typically large keys. Several multivariate schemes participated in the NIST process, and in particular Rainbow [DCK+21] and GeMSS [CFM+21] were a finalist and alternate proposal in Round 3. Unfortunately, both schemes received significant attacks [TPD21, Beu21, Beu22a] during the last round that led to their exclusion. Despite these attacks, the robustness of the multivariate assumptions was not undermined and several signature schemes are still

considered secure [KPG99, Beu20, Beu22b] and are expected to be part of the forthcoming selection process.

In [Beu22b], Beullens introduced the MAYO signature scheme, proposing an innovative technique that takes advantage of the robust construction of Unbalanced Oil and Vinegar (UOV) and drastically reduces the key size while maintaining the efficiency of the original construction. MAYO emerges as a promising scheme, especially in terms of implementation, as it can leverage prior developments associated with multivariate schemes, specifically those pertaining to UOV and Rainbow. However, a thorough analysis of the scheme's performance, particularly on embedded platforms, has yet to be conducted. In this paper, we analyze the implementation of MAYO on ARMv7-M, an architecture for 32-bit processors widely used in consumer electronic devices.

Our contribution In this work, we explore constant-time optimized implementation techniques for ARMv7-M, originally designed for UOV-based schemes, and extend them to the MAYO cryptosystem. The main techniques are extended from [CKY21] on Rainbow and exploit optimized arithmetic on $\mathbb{F}_{16}$ via bitslicing. We describe a new approach for evaluating the polar forms of quadratic maps, which are a key operation in MAYO. Next, we argue how a different parametrization of the scheme can foster greater performance gains, especially on 32-bit architectures. Finally, we analyze the optimized implementation of the scheme with the new parameters on the STM32H753ZIT6 microcontroller, measuring the performance of the signing and verification process. To the best of our knowledge, this is the first optimized implementation of MAYO on the ARM platform.

Source code The source code for our implementation will be available after publication.

Related Work A preliminar implementation of the MAYO scheme for Intel platforms has been made in [Beu22b], using AVX2 instructions. However, this implementation relied on an earlier parameter set based on $\mathbb{F}_{31}$. The MAYO team has since published a draft specification document [BCC+23], which describes the scheme in detail and presents an updated set of parameters that aligns with the approach taken in this work. The performance of the scheme has been evaluated on Intel platforms, with the optional use of AES-NI instructions.

Recent literature on optimizing multivariate schemes for ARM architecture has primarily focused on the Rainbow scheme. Many of the implementation techniques that can be adapted to UOV-based schemes are derived from [CKY21], which targets the Cortex-M4 platform. Following the attacks on Rainbow, a novel parametrization for UOV was proposed in [BCH+23]. This updated parametrization integrates recent advances from the literature, reassessing the competitiveness of UOV also in the context of embedded platforms.

Outline In Section 2 we introduce multivariate quadratic maps and the characteristics of the ARMv7-M architecture. In Section 3 we describe the MAYO signature scheme, discuss its security and its original parametrization. Section 4 introduces the optimized implementation techniques for UOV-based schemes and describes their extension to MAYO. In Section 4.4 new parameters for MAYO are proposed, showing how they can lead to better performance. Finally, in Section 5, we analyze the resulting implementation of MAYO with updated parameters and compare it with the original set.

2 Preliminaries

Notation For $a, b \in \mathbb{N}$, we denote by $[a, b]$ the set $\{a, \dots, b\}$ and similarly we denote by $[b]$ the set $\{1, \dots, b\}$. We write $\mathbb{F}_q$ for a finite field of q elements. We denote by $\mathbb{F}_q^{m \times n}$ the set of matrices over $\mathbb{F}_q$ with m rows and n columns. $\mathbf{I}_{n \times n}$ is the identity matrix of size n. $\mathbf{0}_{m \times n}$ is the $m \times n$ zero matrix. Unless otherwise specified, we denote by $\boldsymbol{x} \in \mathbb{F}_q^n$ a column vector. $\mathbf{0}_n$ is the zero vector in $\mathbb{F}_q^n$. For a sequence of elements $x_1, \dots, x_n \in \mathbb{F}_q$, we denote by $[x_i]_{i \in [n]}$ the column vector $(x_1, \dots, x_n) \in \mathbb{F}_q^n$. For a sequence of vectors $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m \in \mathbb{F}_q^n$, we denote by $[\boldsymbol{x}_i]_{i \in [m]}$ the matrix in $\mathbb{F}_q^{m \times n}$ with $\boldsymbol{x}_i$ as the i-th row. For simplicity, for $\boldsymbol{x} \in \mathbb{F}_q^n, \boldsymbol{y} \in \mathbb{F}_q^m$, we write $(\boldsymbol{x}, \boldsymbol{y})$ as the column vector $(\boldsymbol{x}^\intercal \parallel \boldsymbol{y}^\intercal)^\intercal \in \mathbb{F}_q^{n+m}$ obtained from the concatenation of the two vectors. For a finite set X, we write $x \leftarrow\!\!\$\ X$ to denote the sample of the element x from the uniform distribution over X. Finally, we denote by $\binom{n}{k}$ the binomial coefficient.

Multivariate quadratic maps Multivariate hash-and-sign signature schemes are based on multivariate quadratic trapdoor functions. A *multivariate quadratic map* $\mathcal{P} \colon \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^m$ with m components and n variables is defined by m multivariate quadratic polynomials $p_1(\boldsymbol{x}), \dots, p_m(\boldsymbol{x})$ in n variables $\boldsymbol{x} = (x_1, \dots, x_n)$ with coefficients in a finite field $\mathbb{F}_q$. The evaluation of $\mathcal{P}$ at $\boldsymbol{v} \in \mathbb{F}_q^n$ is $\boldsymbol{t} = (t_1, \dots, t_m) \in \mathbb{F}_q^m$, where $t_i = p_i(\boldsymbol{v})$ for $i = 1, \dots, m$. In the following, we will only consider homogeneous quadratic maps $\mathcal{P}$, for which all components p_i have no linear or constant part.

For a homogeneous multivariate quadratic polynomial p, we can define its *polar form*:

$$b(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x} + \boldsymbol{y}) - p(\boldsymbol{x}) - p(\boldsymbol{y}).$$

Similarly, for a homogeneous multivariate quadratic map $\mathcal{P} = (p_1, \dots, p_m) \colon \mathbb{F}_q^n \to \mathbb{F}_q^m$, its polar form is defined as $\mathcal{B} = (b_1, \dots, b_m) \colon \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q^m$. It can be shown that the polar form of a multivariate quadratic map is a symmetric and bilinear map.

The security of multivariate schemes is strongly linked to the computational hardness of the Multivariate Quadratic (MQ) problem. Let $\mathcal{P} \colon \mathbb{F}_q^n \to \mathbb{F}_q^m$ be a multivariate quadratic function. Given $\boldsymbol{t} \leftarrow\!\!\$\ \mathbb{F}_q^m$, the MQ problem asks to find a preimage $\boldsymbol{v} \in \mathbb{F}_q^n$ such that $\mathcal{P}(\boldsymbol{v}) = \boldsymbol{t}$. The MQ problem is NP-hard over a finite field. Furthermore, it is believed to be hard on average if $n \sim m$, both classically and quantumly. Only exponential-time algorithms are known to solve random instances of the problem for these parameters. If a polynomial system is highly overdetermined or highly underdetermined, there exist special algorithms that run in polynomial time [MHT13, CKPS00].

ARMv7-M The ARMv7-M is a processor architecture designed by ARM targeting embedded microcontrollers. The first CPU supporting this architecture was the Cortex-M3, but the ARMv7-M is fully supported by more recent CPU such as the Cortex-M4, Cortex-M7, Cortex-M23 and Cortex-M33. It leverages on 16 32-bit registers, three of which are reserved for stack pointer, program counter, and link register. It supports Thumb-2 instructions, where bitwise and arithmetic instructions take one cycle (with the exception of the non constant-time `umull` instruction on Cortex-M3 and division). Load, stores, and branches can take multiple cycles. A key feature of the ARMv7-M that we will leverage to achieve constant-time implementations is the availability of conditional execution of instructions that allows to selectively execute instructions based on the value of flags in the Application Program Status Register (APSR), with no impact on the execution time. Some examples are provided in Appendix A.

3 MAYO

The MAYO signature scheme is based on the trapdoor function of the Unbalanced Oil and Vinegar (UOV) signature scheme. In UOV, the trapdoor function is a multivariate quadratic map $\mathcal{P}\colon \mathbb{F}_q^n \to \mathbb{F}_q^m$ that vanishes on a secret linear subspace $O \subset \mathbb{F}_q^n$ of dimension $\dim O = m$.

Given a target $\boldsymbol{y} \in \mathbb{F}_q^m$, the knowledge of the secret O can be used to find a preimage $\boldsymbol{x} \in \mathbb{F}_q^n$ for $\boldsymbol{y}$ by reducing the MQ problem to a linear system. Start by randomly sampling $\boldsymbol{v} \in \mathbb{F}_q^n$ and then solve $\mathcal{P}(\boldsymbol{v} + \boldsymbol{o}) = \boldsymbol{t}$ for $\boldsymbol{o} \in O$. Since

$$\boldsymbol{t} = \mathcal{P}(\boldsymbol{v} + \boldsymbol{o}) = \underbrace{\mathcal{P}(\boldsymbol{v})}_{\text{fixed}} + \underbrace{\mathcal{P}(\boldsymbol{o})}_{=0} + \underbrace{\mathcal{B}(\boldsymbol{v}, \boldsymbol{o})}_{\text{linear in } \boldsymbol{o}}, \tag{1}$$

the system reduces to the linear system $\mathcal{B}(\boldsymbol{v}, \boldsymbol{o}) = \boldsymbol{t} - \mathcal{P}(\boldsymbol{v})$ of m equation and m variables $\boldsymbol{o}$. Notice that whenever the linear map $\mathcal{B}(\boldsymbol{v}, \cdot)$ is non-singular, the system has a unique solution $\boldsymbol{o} \in O$ and the preimage is $\boldsymbol{x} = \boldsymbol{v} + \boldsymbol{o}$.

During key generation, a large part of the public map $\mathcal{P}$ can be chosen at random before choosing O. This makes it possible to significantly reduce the size of the public key, for example, by deterministically expanding a seed, following the approach of [PTBW11]. More precisely, the size of $\mathcal{P}$ is reduced to $m \cdot o(o+1)/2$ plus the length of the seed, where m is the number of polynomials in the quadratic map and o is the dimension of the secret subspace O. By choosing $o < m$ one could randomly generate a larger portion of $\mathcal{P}$, reducing the size of the public key. In addition, the size of O affects the complexity of key recovery attacks on the scheme, so reducing it would allow one to choose overall smaller parameters. However, in UOV it is required that $o = m$, since otherwise the system of (1) would be overdetermined and, with high probability, without a solution.

3.1 Whipping the UOV map

With MAYO, Beullens proposes a *whipping* technique enabling the utilization of a smaller secret subspace O of dimension $\dim O = o < m$. The idea is to start from a multivariate quadratic map $\mathcal{P}\colon \mathbb{F}_q^n \to \mathbb{F}_q^m$ that vanishes on O, and to deterministically transform it into a larger map $\mathcal{P}^*\colon \mathbb{F}_q^{kn} \to \mathbb{F}_q^m$, for some $k > 1$, which vanishes on $O^k \subset \mathbb{F}_q^{kn}$ of dimension $ko \geq m$. The map $\mathcal{P}^*$ is referred to as the *whipping transformation* of $\mathcal{P}$.

In [Beu22b], the whipping transformation is obtained by choosing $k(k+1)/2$ random invertible matrices $\{\mathbf{E}_{i,j} \in \mathrm{GL}_m(\mathbb{F}_q)\}_{1 \leq i \leq j \leq k}$ and defining

$$\mathcal{P}^*(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = \sum_{i=1}^{k} \mathbf{E}_{i,i}\, \mathcal{P}(\boldsymbol{x}_i) + \sum_{1 \leq i < j \leq k} \mathbf{E}_{i,j}\, \mathcal{B}(\boldsymbol{x}_i, \boldsymbol{x}_j).$$

The matrices $\mathbf{E}_{i,j}$ can be part of the system parameters or selected randomly during key generation. In the first case, in the specification it is currently proposed the utilization of $k(k+1)/2 < m$ maps that represent multiplication by X^j. We fix a monic irreducible polynomial $f(X)$ of degree m and consider the extension field $\mathbb{F}_{q^m} = \mathbb{F}[X]/(f(X))$. Then we can set the matrices to be representations of the multiplication by $1, X, X^2, \ldots, X^{\frac{k(k+1)}{2}-1}$ in $\mathbb{F}_{q^m}$. In the following we suppose the matrices $\mathbf{E}_{i,j}$ are part of the system parameters and that are obtained as above.

3.2 Signature scheme description

The parameters of the scheme are the integers q, n, o, m, k such that $ko \geq m$, the matrices $\{\mathbf{E}_{i,j} \in \mathrm{GL}_m(\mathbb{F}_q)\}_{1 \leq i \leq j \leq k}$, and an opportune hash function $\mathsf{H}\colon \{0,1\}^* \to \mathbb{F}_q^m$. The algorithms for key generation, signing, and verification are described in detail in Algorithm 1.

Key Generation Fix a random matrix $\mathbf{O} \in \mathbb{F}_q^{o\times(n-o)}$ and let the secret subspace O be the row space of $(\mathbf{O}\,\mathbf{I}_{o\times o}) \in \mathbb{F}_q^{o\times n}$. This condition does not excessively restrict the key space, as most subspaces of dimension o have this form (for the parameters listed in Table 1 and Table 2). Then, one randomly chooses a quadratic map $\mathcal{P}$ that vanishes on O. Since $\mathcal{P} = (p_1, \ldots, p_m)$ is homogeneous quadratic, each p_i has an upper triangular matrix representation $\mathbf{P}_i \in \mathbb{F}_q^{n\times n}$ such that $\boldsymbol{x}^\intercal \mathbf{P}_i \boldsymbol{x} = p_i(\boldsymbol{x})$. Consider the matrix representation $\mathbf{P}_i$ divided into the following submatrices:

$$\mathbf{P}_i = \begin{pmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0}_{o\times o} & \mathbf{P}_i^{(3)} \end{pmatrix}.$$

The matrices $\mathbf{P}_i^{(1)} \in \mathbb{F}_q^{(n-o)\times(n-o)}$ (upper diagonal) and $\mathbf{P}_i^{(2)} \in \mathbb{F}_q^{(n-o)\times o}$ are randomly chosen by expanding a short seed. The matrix $\mathbf{P}_i^{(3)}$ is subsequently computed by imposing the condition $p_i(O) = \mathbf{0}_m$:

$$(\mathbf{O}\,\mathbf{I}_{o\times o}) \begin{pmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0}_{o\times o} & \mathbf{P}_i^{(3)} \end{pmatrix} (\mathbf{O}\,\mathbf{I}_{o\times o})^\intercal = \mathbf{O}\,\mathbf{P}_i^{(1)}\,\mathbf{O}^\intercal + \mathbf{O}^\intercal\,\mathbf{P}_i^{(2)} + \mathbf{P}_i^{(3)} = \mathbf{0}_m.$$

So one can set $\mathbf{P}_i^{(3)}$ as the upper triangular part of $\mathbf{M} = -\mathbf{O}\,\mathbf{P}_i^{(1)}\,\mathbf{O}^\intercal - \mathbf{O}^\intercal\,\mathbf{P}_i^{(2)}$, that is, the skew-symmetric matrix $\mathbf{M} + \mathbf{M}^\intercal$.

Signature Computation Given a message $m \in \{0,1\}^*$, its signature is computed by sampling a salt $r \leftarrow\!\!\$\ \{0,1\}^\lambda$ and computing a preimage $\boldsymbol{x}$ of $\boldsymbol{t} = \mathsf{H}(m \,\|\, r) \in \mathbb{F}_q^m$. Choose random vectors $(\bar{\boldsymbol{v}}_1, \ldots, \bar{\boldsymbol{v}}_k) \in \mathbb{F}^{kn}$ where the last o components of $\bar{\boldsymbol{v}}_i$ are zero. Then try to find $(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_k) \in O^k$ that solves the system

$$\mathcal{P}^*(\bar{\boldsymbol{v}}_1 + \boldsymbol{o}_1, \ldots, \bar{\boldsymbol{v}}_k + \boldsymbol{o}_k) = \boldsymbol{w}. \tag{2}$$

This system has m linear equations in $ko \geq m$ variables, so it will be solvable with high probability. The signature is the pair $\sigma = (r, \boldsymbol{x})$, where $\boldsymbol{x} = (\bar{\boldsymbol{v}}_1 + \boldsymbol{o}_1, \ldots, \bar{\boldsymbol{v}}_k + \boldsymbol{o}_k) \in \mathbb{F}^{kn}$.

Similarly to Equation (1) for UOV, Equation (2) can be rewritten as

$$\begin{aligned}\boldsymbol{t} &= \sum_{s=1}^{k} \mathbf{E}_{i,i}\,\mathcal{P}(\bar{\boldsymbol{v}}_i + \boldsymbol{o}_i) + \sum_{1\leq i<j\leq k} \mathbf{E}_{i,j}\,\mathcal{B}(\bar{\boldsymbol{v}}_i + \boldsymbol{o}_i, \bar{\boldsymbol{v}}_j + \boldsymbol{o}_j) \\ &= \sum_{i=1}^{k} \mathbf{E}_{i,i}\big(\mathcal{P}(\bar{\boldsymbol{v}}_i) + \mathcal{B}(\bar{\boldsymbol{v}}_i, \boldsymbol{o}_i)\big) + \sum_{1\leq i<j\leq k} \mathbf{E}_{i,j}\big(\mathcal{B}(\bar{\boldsymbol{v}}_i, \bar{\boldsymbol{v}}_j) + \mathcal{B}(\bar{\boldsymbol{v}}_i, \boldsymbol{o}_j) + \mathcal{B}(\bar{\boldsymbol{v}}_j, \boldsymbol{o}_i)\big).\end{aligned}$$

The constant term of the system is given by $\boldsymbol{t} - \boldsymbol{w}$, where

$$\boldsymbol{w} = \sum_{i=1}^{k} \mathbf{E}_{i,i}\,\mathcal{P}(\bar{\boldsymbol{v}}_i) + \sum_{1\leq i<j\leq k} \mathbf{E}_{i,j}\,\mathcal{B}(\bar{\boldsymbol{v}}_i, \bar{\boldsymbol{v}}_j) \in \mathbb{F}^m.$$

Notice that for $\bar{\boldsymbol{v}} \in \mathbb{F}_q^n$ of the form $(\boldsymbol{v}, \mathbf{0}_o)$, to evaluate $\mathcal{P}(\bar{\boldsymbol{v}})$ only the $\mathbf{P}_i^{(1)}$ component of p_i is required, so that $p_i(\bar{\boldsymbol{v}}) = \boldsymbol{v}^\intercal \mathbf{P}_i^{(1)} \boldsymbol{v}$. Similarly, the evaluation of the polar form $\mathcal{B} = (b_1, \ldots, b_m)$ on a pair $(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}})$ of the form $(\boldsymbol{u}, \mathbf{0}_o), (\boldsymbol{v}, \mathbf{0}_o)$ can be computed as $b_i(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}) = \boldsymbol{u}^\intercal \left(P_i^{(1)} + (P_i^{(1)})^\intercal\right) \boldsymbol{v}$.

The linear part of the system is

$$\sum_{i=1}^{k} \mathbf{E}_{i,i}\,\mathcal{B}(\bar{\boldsymbol{v}}_i, \boldsymbol{o}_i) + \sum_{i<j} \mathbf{E}_{i,j}\big(\mathcal{B}(\bar{\boldsymbol{v}}_i, \boldsymbol{o}_j) + \mathcal{B}(\bar{\boldsymbol{v}}_j, \boldsymbol{o}_i)\big).$$

Table 1: Original parameters of MAYO [Beu22b]

Security level	n	m	o	k	q
I	66	67	5	14	16
III	98	99	6	17	16
V	130	132	7	19	16

The linear part in the signature generation process is computed through linear transformations $\mathcal{L}_{\bar{\boldsymbol{v}}} : \mathcal{B}(\bar{\boldsymbol{v}}, \cdot) : O \to \mathbb{F}_q^m$, with $\bar{\boldsymbol{v}}$ of the form $(\boldsymbol{v}, \mathbf{0}_o)$. For a vector $\boldsymbol{c} \in \mathbb{F}_q^o$, consider $\boldsymbol{o} = (\mathbf{O}\,\mathbf{I}_{o\times o})^\intercal \boldsymbol{c} \in O$. Then, each component $b_i(\bar{\boldsymbol{v}}, \boldsymbol{o})$ of $\mathcal{L}_{\bar{\boldsymbol{v}}}(\boldsymbol{o})$ can be written as

$$\begin{aligned} b_i(\bar{\boldsymbol{v}}, \boldsymbol{o}) &= (\boldsymbol{v}, \mathbf{0}_o)^\intercal \begin{pmatrix} \mathbf{P}_i^{(1)} + (\mathbf{P}_i^{(1)})^\intercal & \mathbf{P}_i^{(2)} \\ (\mathbf{P}_i^{(2)})^\intercal & \mathbf{P}_i^{(3)} + (\mathbf{P}_i^{(3)})^\intercal \end{pmatrix} (\mathbf{O}\,\mathbf{I}_{o\times o})^\intercal \boldsymbol{c} \\ &= \boldsymbol{v}^\intercal \left((\mathbf{P}_i^{(1)} + (\mathbf{P}_i^{(1)})^\intercal)\,\mathbf{O}^\intercal + \mathbf{P}_i^{(2)} \right) \boldsymbol{c}. \end{aligned}$$

It follows that linear maps $\mathcal{L}_{\bar{\boldsymbol{v}}}$ can be represented by using matrices $\mathbf{L}_i = (\mathbf{P}_i^{(1)} + (\mathbf{P}_i^{(1)})^\intercal)\,\mathbf{O}^\intercal + \mathbf{P}_i^{(2)}$ which satisfy

$$\mathcal{L}_{\bar{\boldsymbol{v}}}(\boldsymbol{o}) = \{\boldsymbol{v}^\intercal \mathbf{L}_i \boldsymbol{c}\}_{i\in[m]}.$$

Signature Verification Given a signature $\sigma = (r, \boldsymbol{x})$ for a message m, compute $\boldsymbol{t}' = \mathcal{P}^*(\boldsymbol{x}) \in \mathbb{F}_q^m$. If $\boldsymbol{t}' = \mathsf{H}(m \,\|\, r)$ accept, otherwise reject.

3.3 Security and original parameters

As MAYO represents a novel cryptographic proposal, to date, no instances of attacks have been reported that specifically target its structure. Current understanding of potential key recovery attacks on MAYO is derived from previous research on the UOV scheme, with the sole variation being the choice of $o < m$. However, it is important to note that while the foundational Oil and Vinegar construction has been extensively analyzed and is widely regarded as secure, the addition of the whipping structure could potentially hide exploitable vulnerabilities.

In the following, we summarize the best known attacks against UOV. The descriptions are generalized for the case where $o \leq m$, as it applies to MAYO, and their presentation closely follows that given in [Beu22b]. The original parameters proposed in [Beu22b] are given in Table 1.

Direct Attack In a direct approach, an attacker tries to forge a signature $\mathbf{x}$ for an hashed message $\boldsymbol{t}$ by directly solving the polynomial system $\mathcal{P}^*(\boldsymbol{x}) = \boldsymbol{t}$. A direct attack ignores the structure of the scheme, so it can be used to attack any multivariate cryptosystem. In the case of MAYO, we, therefore, consider $\mathcal{P}^*(\boldsymbol{x}) = \mathsf{H}(r \,\|\, m)$ as a generic instance of the MQ problem.

The main tool to solve nonlinear polynomial systems over finite fields is based on the computation of Gröbner bases. The best known algorithms are Faugere's F4/F5 [Fau02], XL [CKPS00], or the Hybrid Approach [BFP09]. The complexity of Gröbner basis algorithms has been the object of extensive studies, in the following and in the security estimates for the new parameters given in Section 4.4, we refer to [BMSV22] for an estimate of its complexity.

The whipped quadratic map $\mathcal{P}^*\colon \mathbb{F}_q^{kn} \to \mathbb{F}_q^m$ yields systems of m equations and kn variables. These systems are highly underdetermined, therefore it is possible to apply

Algorithm 1: MAYO Signature Scheme

$\mathsf{KGen}(1^\lambda)$:

$\mathbf{O} \leftarrow\$ \mathbb{F}_q^{o\times(n-o)}$
for $i \leftarrow 1,\dots,m$ **do**
  $\mathbf{P}_i^{(1)} \leftarrow\$ \mathbb{F}_q^{(n-o)\times(n-o)}$ ▷ Upper triangular
  $\mathbf{P}_i^{(2)} \leftarrow\$ \mathbb{F}_q^{(n-o)\times o}$
  $\mathbf{P}_i^{(3)} \leftarrow \mathsf{Upper}(-\mathbf{O}\,\mathbf{P}_i^{(1)}\,\mathbf{O}^\intercal - \mathbf{O}^\intercal\,\mathbf{P}_i^{(2)})$ ▷ $\mathsf{Upper}(\mathbf{M}) \leftarrow \mathbf{M} + \mathbf{M}^\intercal$
  $\mathbf{P}_i \leftarrow \begin{pmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0}_{o\times o} & \mathbf{P}_i^{(3)} \end{pmatrix}$
  $\mathbf{L}_i \leftarrow (\mathbf{P}_i^{(1)} + (\mathbf{P}_i^{(1)})^\intercal)\,\mathbf{O}^\intercal + \mathbf{P}_i^{(2)}$
$\mathsf{pk} \leftarrow \{\mathbf{P}_i\}_{i\in[m]}$
$\mathsf{sk} \leftarrow (\mathbf{O}, \{\mathbf{P}_i^{(1)}\}_{i\in[m]}, \{\mathbf{L}_i\}_{i\in[m]})$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{Sign}(\mathsf{sk}, m)$:

$(\mathbf{O}, \{\mathbf{P}_i^{(1)}\}_{i\in[m]}, \{\mathbf{L}_i\}_{i\in[m]}) \leftarrow \mathsf{sk}$
$\boldsymbol{w} \leftarrow \mathbf{0}_m$
$\mathbf{A} \leftarrow (\mathbf{A}_1,\dots,\mathbf{A}_k) \leftarrow (\mathbf{0}_{m\times o},\dots,\mathbf{0}_{m\times o})$
repeat
  $r \leftarrow\$ \{0,1\}^\lambda$
  $\boldsymbol{t} \leftarrow \mathsf{H}(m \,\|\, r)$
  for $i \leftarrow 1,\dots,k$ **do**
    $\boldsymbol{v}_i \leftarrow\$ \mathbb{F}_q^{n-o}$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \mathbf{E}_{i,i}\left[\boldsymbol{v}_i^\intercal \mathbf{P}_t^{(1)} \boldsymbol{v}_i\right]_{t\in[m]}$
    $\mathbf{A}_i \leftarrow \mathbf{A}_i + \mathbf{E}_{i,i}\left[\boldsymbol{v}_i^\intercal \mathbf{L}_t\right]_{t\in[m]}$
    for $j \leftarrow i+1,\dots,k$ **do**
      $\boldsymbol{v}_j \leftarrow\$ \mathbb{F}_q^{n-o}$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \mathbf{E}_{i,j}\left[\boldsymbol{v}_i^\intercal (\mathbf{P}_t^{(1)} + (\mathbf{P}_t^{(1)})^\intercal)\boldsymbol{v}_j\right]_{t\in[m]}$
      $\mathbf{A}_i \leftarrow \mathbf{A}_i + \mathbf{E}_{i,j}\left[\boldsymbol{v}_j^\intercal \mathbf{L}_t\right]_{t\in[m]}$
      $\mathbf{A}_j \leftarrow \mathbf{A}_j + \mathbf{E}_{i,j}\left[\boldsymbol{v}_i^\intercal \mathbf{L}_t\right]_{t\in[m]}$
until the system $\mathbf{A}\boldsymbol{c} = \boldsymbol{t} - \boldsymbol{w}$ has a solution $\boldsymbol{c} \in \mathbb{F}_q^{ko}$
$(\boldsymbol{c}_1,\dots,\boldsymbol{c}_k) \leftarrow \boldsymbol{c}$
$\boldsymbol{o} \leftarrow \left((\mathbf{O}\,\mathbf{I}_{o\times o})^\intercal \boldsymbol{c}_1,\dots,(\mathbf{O}\,\mathbf{I}_{o\times o})^\intercal \boldsymbol{c}_k\right)$
$\boldsymbol{x} \leftarrow (\boldsymbol{v}_1 \| \mathbf{0}_o + \boldsymbol{o}_1,\dots,\boldsymbol{v}_k \| \mathbf{0}_o + \boldsymbol{o}_k)$
return $\sigma = (r, \boldsymbol{x})$

$\mathsf{Verify}(\mathsf{pk}, m, \sigma)$:

$(r, \boldsymbol{x}) \leftarrow \sigma$
$\{\mathbf{P}_i\}_{i\in[m]} \leftarrow \mathsf{pk}$
$\boldsymbol{t} \leftarrow \mathsf{H}(m \,\|\, r)$
$\boldsymbol{t}' \leftarrow \mathbf{0}_m$
for $i \leftarrow 1,\dots,k$ **do**
  $\boldsymbol{t}' \leftarrow \boldsymbol{t}' + \mathbf{E}_{i,i}\left[\boldsymbol{x}_i^\intercal \mathbf{P}_t\right]_{t\in[m]}$
  for $j \leftarrow i+1,\dots,k$ **do**
    $\boldsymbol{t}' \leftarrow \boldsymbol{t}' + \mathbf{E}_{i,j}\left[\boldsymbol{x}_i^\intercal (\mathbf{P}_t + (\mathbf{P}_i)^\intercal)\boldsymbol{x}_j\right]_{t\in[m]}$
return $\boldsymbol{t}' = \boldsymbol{t}$

the technique of [TW12] and reduce them to determined systems with $m - \lceil kn/m \rceil + 1$ equations. In the following, we denote by $\mathsf{Direct}_{\mathrm{MQ}}(n, m, q)$ the complexity of solving a quadratic system over $\mathbb{F}_q$ of m equations and n variables. For MAYO, the complexity of the Direct Attack can be estimated by

$$\mathsf{Direct}_{\mathrm{MAYO}}(n, m, o, k, q) = \mathsf{Direct}_{\mathrm{MQ}}(kn, m, q).$$

Kipnis-Shamir Attack The original work on the Oil and Vinegar (OV) scheme by Patarin [Pat97], was attacked by Kipnis and Shamir [KS98]. The attack aims to efficiently recover an equivalent private key for $(\mathcal{P}, O)$ by finding the secret subspace O. The vulnerability is based on a common property of schemes based on the Oil and Vinegar construction: consider $\mathbf{B}_i$ the matrix forms associated with the m components of $\mathcal{B}$. Then, we have that $\mathbf{B}_i O \subset O^\perp$, where $O^\perp$ is the orthogonal component of O. Since we assume that $n = 2o$, both O and $O^\perp$ have the same dimension o. Therefore, whenever $\mathbf{B}_i$ is non singular, we have that $\mathbf{B}_i O = O^\perp$ and O is a common invariant subspace of $\mathbf{B}_i^{-1}\mathbf{B}_j$ for any couple of invertible $\mathbf{B}_i, \mathbf{B}_j$. The search for O then reduces to finding the common invariant subspace of a large set of linear maps, whose complexity can be estimated by $\mathcal{O}(n^3)$.

In later iterations of the scheme, such as UOV and MAYO, we have $n > 2o$. Therefore, the equality $\mathbf{B}_i O = \mathbf{B}_j O$ might no longer hold and O might not be an invariant subspace of $\mathbf{B}_i^{-1}\mathbf{B}_j$. However, it is possible to show that the probability of $\mathbf{B}_i^{-1}\mathbf{B}_j$ having a nontrivial inviariant subspace that is also a subspace of O is approximately q^{2o-n+1}. The complexity of the general attack can be estimated by

$$\mathsf{KS}_{\mathrm{MAYO}}(n, m, o, k, q) = \mathcal{O}\big(q^{n-2o-1} n^4\big).$$

Reconciliation Attack The Reconciliation Attack attempts to find the secret subspace O by solving the system $\mathcal{P}(\boldsymbol{o}) = \mathbf{0}_m$ for $\boldsymbol{o} \in O$. Once an element $\boldsymbol{o}_1 \in O$ is found, the search for another element $\boldsymbol{o}_2 \in O$ is easier because it is possible to impose m linear constraints given by the equations $\mathcal{B}(\boldsymbol{o}_1, \boldsymbol{o}_2)$. The process can be iterated until all o elements of a basis for O are found. Therefore, the complexity of the attack is dominated by the search for the first element $\boldsymbol{o}_1$.

Since O has dimension o, to solve $\mathcal{P}(\boldsymbol{o}_1) = \mathbf{0}_m$, we can impose o linear constraints on $\boldsymbol{o}_1$, obtaining a quadratic system of m equations and $n - o$ variables. If $n - o > m$, the system has many solutions that do not belong to O, so that the search must be repeated numerous times, making the attack inefficient. However, if $n - o \leq m$, the system $\mathcal{P}(\boldsymbol{o}_1) = \mathbf{0}_m$ is expected to have a unique solution in O and the complexity of the attack reduces to the solving of a random system of m quadratic equations in $n - o$ variables, possibly outperforming a direct attack.

$$\mathsf{Recon}_{\mathrm{MAYO}}(n, m, o, k, q) = \mathsf{Direct}_{\mathrm{MQ}}(n - o, m, q).$$

Intersection Attack The Intersection Attack is a generalization of the Reconciliation Attack introduced in [Beu21]. Similarly to Kipnis-Shamir, the attack tries to find simultaneously $k \geq 2$ elements $\boldsymbol{o}_i \in O$ by searching for non-trivial intersections of k subspaces of the form $\mathbf{B}_i O$. If such an intersection is found, then the attack reduces to solving a system of $\binom{k+1}{2} m - 2\binom{k}{2}$ quadratic equations in $\min(n, nk - (2k-1)o)$ variables.

When o is chosen such that $n \geq 3o$, the intersection is not guaranteed to be non-trivial (even with $k = 2$) and the procedure must be repeated with a different choice of subspaces. The probability of finding a non-trivial intersection is approximately $q^{-n+3o-1}$ and the attack must be repeated an average of q^{n-3o+1} times. In the context of MAYO, o is very small compared to n and the attack is much less efficient with an estimated complexity of

$$\mathsf{Inters}_{\mathrm{MAYO}}(n, m, o, k, q) = q^{n-3o+1}\mathsf{Direct}_{\mathrm{MQ}}(n, 3m-2, q).$$

4 Implementation techniques

In this section, we describe the implementation techniques employed in key components of Oil and Vinegar-based schemes. The main techniques are borrowed from the Rainbow reference implementation on ARM Cortex M4 [CKY21]. This implementation can serve as a useful reference for Oil and Vinegar-based schemes, such as MAYO, as most computations remain largely the same.

We focus on the case where the base field is $\mathbb{F} = \mathbb{F}_{16}$, which can benefit from fast bitsliced implementation and will be used in the parametrization proposed in Section 4.4. We are using the representation $\mathbb{F}_{16} = \mathbb{F}_2[X]/(g(X))$ with $g(X) = X^4 + X + 1$, which allows for the most efficient implementation of bitsliced multiplications.

4.1 Bitsliced multiplication of $\mathbb{F}_{16}$ elements

For $a \in \mathbb{F}_{16} = \mathbb{F}_2[X]/(g(X))$, write $a = a_3X^3 + a_2X^2 + a_1X + a_0$ with $a_i \in \mathbb{F}_2$. Any field element a can be represented with four bits packed into a nibble with a_0 at the least significant bit position. Two $\mathbb{F}_{16}$ elements are packed into a byte with the least significant nibble in the lower half of the byte. Addition and subtraction of these elements is a simple bit-wise XOR operation. For multiplication, 32-bit embedded CPUs do not offer carryless multiplication operations. In general, it is more convenient to bitslice more field elements into multiple registers and rely on bitwise operations.

Multiplication over $\mathbb{F}_{16}$ Let $a, b \in \mathbb{F}_{16} = \mathbb{F}_2[X]/(X^4 + X + 1)$, where $a = (a_3, a_2, a_1, a_0)$ and $b = (b_3, b_2, b_1, b_0)$. We can compute the product $c = a \cdot b = (c_3, c_2, c_1, c_0)$ as follows:

$$\begin{aligned} c_3 &= (a_0 + a_3)b_3 + a_1b_2 + a_2b_1 + a_3b_0 \\ c_2 &= (a_2 + a_3)b_3 + (a_0 + a_3)b_2 + a_1b_1 + a_2b_0 \\ c_1 &= (a_1 + a_2)b_3 + (a_2 + a_3)b_2 + (a_0 + a_3)b_1 + a_1b_0 \\ c_0 &= a_1b_3 + a_2b_2 + a_0b_0 + a_3b_1 \end{aligned} \tag{3}$$

However, multiplying elements one by one does not take full advantage of the 32-bit architecture. As we mentioned previously, it is more convenient to simultaneously multiply more elements by a single element whenever possible.

Suppose that we need to multiply exactly 32 field elements by another field element b. We denote with $a_{i,j}$ the jth component of the i-th field element, for $j = 0, \ldots, 3$ and $i = 0, \ldots, 31$. The factor b is organized in the least significant nibble of a register `B`. Elements a_i are stored in four registers `A0`, `A1`, `A2`, `A3` in bitsliced representation. Switching from standard representation to bitsliced one and vice versa can be implemented efficiently in 28 clock cycles, as shown in Algorithm 7.

Algorithm 2 describes the efficient implementation of bitsliced multiplication, following Equation 3. The algorithm can be implemented in 27 clock cycles (Algorithm 8). Algorithm 2 relies on conditional operations which allow us to spare some additions whenever a component of b is null. The constant-time conditional execution offered by the ARMv7-M architecture ensures that bitsliced multiplications are performed in constant time (see Appendix A).

The efficiency of bitsliced multiplications is maximized when we need to multiply 32 elements or multiples of 32. In general, multiplying m elements requires applying the bitslicing algorithm $\lceil m/32 \rceil$ times to bitslice the elements in $\lceil m/8 \rceil$ registers. After that, we apply Algorithm 2 $\lceil m/32 \rceil$ times.

Algorithm 2: Bitsliced multiply-accumulate of 32 $\mathbb{F}_{16}$ elements

Input: 32 bitsliced elements in registers `A0`, `A1`, `A2`, `A3`
Input: Single factor in register `B`
Input: Registers `C0`, `C1`, `C2`, `C3`
Output: Bitsliced result of the multiplication in registers `C0`, `C1`, `C2`, `C3`

```
T0 ← A0 + A3
T1 ← A2 + A3
T2 ← A1 + A2

if b0 ≠ 0 then
    C0 += A0
    C1 += A1
    C2 += A2
    C3 += A3
if b1 ≠ 0 then
    C0 += A3
    C1 += T0
    C2 += A1
    C3 += A2
if b2 ≠ 0 then
    C0 += A2
    C1 += T1
    C2 += T0
    C3 += A1
if b3 ≠ 0 then
    C0 += A1
    C1 += T2
    C2 += T1
    C3 += T0
return C0, C1, C2, C3
```

Algorithm 3: Batched vector-matrix multiplication

Input: $\mathbf{M} \in \mathbb{F}^{n_r n_c \times m}$, representing $\{\mathbf{M}^{(k)} \in \mathbb{F}^{n_r \times n_c}\}_{k \in [m]}$
Input: $\boldsymbol{x} \in \mathbb{F}^{n_r}$
Output: $(\boldsymbol{x}^T \cdot \mathbf{M}^{(1)}, \ldots, \boldsymbol{x}^T \cdot \mathbf{M}^{(k)}) \in \mathbb{F}^m$

```
res ← 0_{m×n_c}
for j ← 1, ..., n_c do
    for i ← 1, ..., n_r do
        [res_{k,j}]_{k∈[m]} +← x_i · [M_{i,j,k}]_{k∈[m]}      ▷ bitsliced multiplication
return res
```

Batched matrix-vector multiplication In Section 3.2, we have seen that multivariate polynomial evaluation can be computed by matrix-vector multiplications. As a result, both private and public operations in multivariate cryptosystems are heavily dependent on finite field multiplications, among which many can be performed in parallel. Bitslicing multiplications can be exploited to efficiently compute batched matrix-vector multiplications, which arise, for example, during the MAYO signing process when calculating $\mathcal{B}(\boldsymbol{v}, \boldsymbol{o})$. This involves multiplying the vector $\boldsymbol{v}$ by all the m linear matrices $L_1, \ldots, L_m$.

Consider m rectangular matrices $\mathbf{M}^{(1)}, \ldots, \mathbf{M}^{(m)}$ of dimension $n_r \times n_c$ to be multiplied for a vector $\boldsymbol{x} \in \mathbb{F}^{n_r}$. To facilitate sequential access to the elements, the matrices are combined into a larger matrix $\mathbf{M} \in \mathbb{F}^{n_r n_c \times m}$, where the k-th row of $\mathbf{M}$ stores $\mathbf{M}^{(k)}$ in row-major order. For clarity, the entry of $\mathbf{M}$ corresponding to $\mathbf{M}^{(k)}_{i,j}$ is denoted as $\mathbf{M}_{i,j,k}$. $\mathbf{M}$ is stored in column-major order. Algorithm 3 illustrates the multiplication process.

To reduce computational burden in this use case, an option is to store the private matrices $L^{(i)}$ in bitslice form, incorporating precomputation into key generation. However, this would render the generated secret keys incompatible with other platforms that do not utilize bitsliced multiplication. Therefore, it is assumed that all keys are stored in standard representation, with elements bitsliced as needed for multiplication, and subsequently converted back to their original form.

Algorithm 4: Constant-time Gaussian elimination

Input: $\mathbf{A} \in \mathbb{F}^{m \times m}$
Input: $\boldsymbol{y} \in \mathbb{F}^{m}$
Output: `fail` $\in \{0, 1\}$, 1 if the system is not solvable
Output: $\boldsymbol{x} \in \mathbb{F}^{m}$ such that $\mathbf{A}\,\boldsymbol{x} = \boldsymbol{y}$

1: $\mathbf{A}' \leftarrow (\mathbf{A}\,\boldsymbol{y}) \in \mathbb{F}^{m \times (m+1)}$
2: `fail` $\leftarrow 0$
3: **for** $i \leftarrow 1, \ldots, m$ **do**
4: **for** $j \leftarrow i+1, \ldots, m$ **do**
5: $p \leftarrow \mathbf{A}'_{i,i}$
6: **for** $k \leftarrow i, \ldots, m+1$ **do**
7: **if** $p = 0$ **then** $\mathbf{A}'_{i,k} \leftarrow \mathbf{A}'_{i,k} + \mathbf{A}'_{j,k}$
8: **if** $\mathbf{A}'_{i,i} = 0$ **then** `fail` $\leftarrow 1$
9: $p^{-1} \leftarrow \mathbf{A}'^{-1}_{i,i}$ ▷ constant look-up table inversion
10: $\left[\mathbf{A}'_{i,k}\right]_{k \in [i,m+1]} \leftarrow p^{-1} \cdot \left[\mathbf{A}'_{i,k}\right]_{k \in [i,m+1]}$ ▷ bitsliced multiplication
11: **for** $j = 1, \ldots, m$ **do**
12: **if** $j = i$ **then continue**
13: $t \leftarrow \mathbf{A}'_{j,i}$
14: $\left[\mathbf{A}'_{j,k}\right]_{k \in [i,m+1]} \mathrel{-}\!\leftarrow t \cdot \left[\mathbf{A}'_{i,k}\right]_{k \in [i,m+1]}$ ▷ bitsliced multiplication
15: $(\mathbf{I}_{m \times m}, \boldsymbol{x}) \leftarrow \mathbf{A}'$
16: **return** $\boldsymbol{x}$, `fail`

4.2 Constant-time Gaussian Elimination

Solving linear systems constitutes a significant component in the signing process of a UOV-based signature scheme, typically achieved through Gaussian elimination. However, as this method operates on confidential inputs, it is imperative that the implementation is constant-time, which is not inherently the case for Gaussian elimination. This constraint arises from the need to examine all pivot elements and replace rows with null pivots prior to executing the elimination process, which may lead to time leakage.

Suppose we have a determined linear system of m equations represented by the matrix $\mathbf{A} \in \mathbb{F}^{m \times m}$, and we want to find a solution to the system $\mathbf{A}\boldsymbol{x} = \boldsymbol{y}$. For the implementation of Rainbow on Cortex-M4, in [CKY21] the authors employ a matrix inversion algorithm inspired by the constant-time Gaussian elimination approach originally introduced in [BCS13], followed by a multiplication by $\boldsymbol{y}$. Since in the application to MAYO it is only required to find a solution for the system, we can avoid the explicit calculation of $\mathbf{A}^{-1}$ and solve for $\boldsymbol{x}$ in $\mathbf{A}\boldsymbol{x} = \boldsymbol{y}$. The procedure is described in detail in Algorithm 4.

For the efficient computation of the inverse of an element in $\mathbb{F}_{16}$ we use a constant-time look-up table. In Algorithm 9 we precompute the 8 bytes look-up table and sequentially load it into a temporary register. We load two bytes of the look-up table at a time using four `movw` instructions and perform shifting operations to select the right bits.

The multiplications on Lines 10, 14 are performed bitsliced. Although this can be extended to the entire algorithm, it is not convenient to store the augmented matrix $\mathbf{A}'$ in bitsliced representation. The main reason is that individual access to pivot elements is required during the process, which would be inefficient if the matrix were in bitsliced form. In details, on Line 10, $m+1$ elements are first bitsliced and then multiplied by p^{-1}. The resulting bitsliced values are stored in temporary registers, avoiding the need to recompute the same elements on Line 14. Subsequently, on Line 14, further bitsliced multiplications are performed, then these values must be unbitsliced. Overall, the algorithms require to bitslice $m+1$ elements a total of $m(1+m)$ times, and multiply $m+1$ bitsliced elements.

Algorithm 5: Evaluation of MQ map

Input: Macaulay matrix $\mathbf{M} \in \mathbb{F}^{m \times \binom{n+1}{2}}$ representing a MQ map $\mathcal{P}$
Input: $\boldsymbol{x} \in \mathbb{F}^n$
Output: $\boldsymbol{y} = \mathcal{P}(\boldsymbol{x}) \in \mathbb{F}^m$

EvalConst($\mathbf{M}$, $\boldsymbol{x}$):
1: $\boldsymbol{y} \leftarrow \mathbf{0}_m$
2: **for** $i \leftarrow 1, \ldots, n$ **do**
3: $\quad \text{acc} \leftarrow \mathbf{0}_m$
4: $\quad$ **for** $j \leftarrow i, \ldots, n$ **do**
5: $\quad\quad \left[\text{acc}_k\right]_{k\in[m]} \mathrel{+}\!\leftarrow x_j \cdot \left[\mathbf{M}_{i,j,k}\right]_{k\in[m]}$ ▷ bitsliced multiplication
6: $\quad \left[y_k\right]_{k\in[m]} \mathrel{+}\!\leftarrow x_i \cdot \left[\text{acc}_k\right]_{k\in[m]}$ ▷ bitsliced multiplication
7: **return** $\boldsymbol{y}$

EvalLeak($\mathbf{M}$, $\boldsymbol{x}$):
1: $\text{acc} \leftarrow \mathbf{0}_{(q-1)\times m}$
2: **for** $i \leftarrow 1, \ldots, n$ **do**
3: $\quad$ **if** $x_i = 0$ **then continue**
4: $\quad$ **for** $j \leftarrow i, \ldots, n$ **do**
5: $\quad\quad$ **if** $x_j = 0$ **then continue**
6: $\quad\quad t \leftarrow x_i\, x_j \in \mathbb{F}$ ▷ look-up table
7: $\quad\quad$ **for** $k \leftarrow 1 \ldots, m$ **do**
8: $\quad\quad\quad \text{acc}_{t,k} \leftarrow \text{acc}_{t,k} + \mathbf{M}_{i,j,k}$
9: $\boldsymbol{y} \leftarrow \text{acc}_1 \in \mathbb{F}^m$
10: **for** $t \leftarrow \mathbb{F} \setminus \{0, 1\}$ **do**
11: $\quad \left[y_k\right]_{k\in[m]} \mathrel{+}\!\leftarrow t \cdot \left[\text{acc}_{t,k}\right]_{k\in[m]}$ ▷ bitsliced multiplication
12: **return** $\boldsymbol{y}$

Gaussian elimination for underdetermined systems For the MAYO signing process, we need to solve the linear system of Equation (2), which is an underdetermined system of m quadratic equations in $ko > m$ variables. To apply the constant-time Gaussian elimination algorithm, we modify Algorithm 4 so that it takes as input matrix $\mathbf{A} \in \mathbb{F}^{m\times n}$ with $n \geq m$. The algorithm proceeds similarly, but the augmented matrix on Line 1 is obtained from a subset of the columns of $\mathbf{A}$. First, an ordered set J of m indices is chosen from $[n]$. Then the Gaussian elimination process is performed on $(\mathbf{A}_{i,j})_{i\in[m],j\in J} \in \mathbb{F}^{m\times m}$. If the algorithm returns a solution $\boldsymbol{x'} \in \mathbb{F}^m$, we return $\boldsymbol{x} \in \mathbb{F}^n$ such that

$$x_j = \begin{cases} x'_j & \text{if } j \in J \\ 0 & \text{otherwise} \end{cases}.$$

If the restricted system is not solvable, the process is restarted with a new choice of salt r. The probability that the restricted determined systems are not solvable is approximately $1/16$, therefore we expect to find a solution after a few tries.

4.3 Evaluating Multivariate Quadratic maps

The evaluation of multivariate quadratic maps is crucial in MPKC signatures, where the public quadratic map $\mathcal{P}$ needs to be directly evaluated for verification. Moreover, in UOV-based schemes, this operation is also required, in constant time, during the signing process. We then proceed to describe two algorithms for the evaluation, in constant and variable time, of $\mathcal{P}$, as illustrated in Algorithm 5

Let $\mathcal{P} = (p_1, \ldots, p_m)$ be a quadratic system of m equations in n variables, where $p_k(\boldsymbol{x}) = \sum_{1\leq i\leq j\leq n} a_{i,j}^{(k)} x_i x_j$ for $k = 1, \ldots, m$. $\mathcal{P}$ can be stored as a Macaulay matrix $\mathbf{M} \in \mathbb{F}^{m\times\binom{n+1}{2}}$. $\mathbf{M}$ is stored in column-major form for sequential loading of coefficients during the computations and we denote with $\mathbf{M}_{i,j,k}$ the entry of $\mathbf{M}$ corresponding to the coefficient $a_{i,j}^{(k)}$.

To achieve private evaluation, we want to avoid direct computation of singular monomials $x_i x_j$, since these multiplications are tedious to bitslice. This involves first computing the multiplications by x_j and storing the result in a separate accumulator, followed by the

multiplications by x_i. We thus describe $p_k(\boldsymbol{x})$ as

$$p_k(\boldsymbol{x}) = \sum_{i=1}^{n} \Big(\sum_{j=i}^{n} a_{i,j}^{(k)} x_j \Big) x_i. \tag{4}$$

Storing $\mathbf{M}$ in column-major form allows the sequential loading of coefficients to be multiplied by x_j. Overall, constant-time evaluation requires to bitslice and multiply m elements $n(n+1)/2 + m$ times.

In the public signature verification process, the evaluation can be significantly accelerated by leveraging variable-time operations, as demonstrated in [CKY21, Section 3.3]. The main difference from the constant-time case involves the direct evaluation of the product $x_i x_j$ using a look-up table of 256 elements. We store an accumulator of m elements for each non-zero field element and sequentially add the columns of $\mathbf{M}$ to the row of the accumulator corresponding to the value of $x_i x_j$. This operation, although highly efficient, is unsuitable for private operations since it would leak the value of the monomial. The result is obtained by bitslicing the accumulator rows and multiplying each with the corresponding field element. Overall, variable-time evaluation requires to bitslice and multiply m elements $q - 2 = 14$ times. This results in a markedly more efficient public evaluation than private ones, at the expense of $(q-1)m\lceil log_2(q) \rceil$ bits of memory. However, for $\mathbb{F}_{16}$, only additional 512 bytes are required.

Evaluation of Polar Forms MAYO also requires the evaluation of polar forms $\mathcal{B}$ in both the signing and verification processes. We could trivially obtain the evaluation of $\mathcal{B}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{P}(\boldsymbol{x} + \boldsymbol{y}) - \mathcal{P}(\boldsymbol{x}) - \mathcal{P}(\boldsymbol{y})$ by performing three evaluations of the public map $\mathcal{P}$ and then adding the results. However, it is possible to perform the computation in a more efficient and compact way, as illustrated in Algorithm 6.

Let $p(\boldsymbol{x}) = \sum_{i \leq j} a_{i,j} x_i x_j$, the polar form of p is given by

$$\begin{aligned} b(\boldsymbol{x}, \boldsymbol{y}) &= \sum_{i \leq j} a_{i,j} \Big[(x_i + y_i)(x_j + y_j) - x_i x_j + y_i y_j \Big] \\ &= \sum_{i \leq j} a_{i,j} (x_i y_j + y_i x_j). \end{aligned} \tag{5}$$

Therefore, for public evaluation, it is sufficient to slightly modify the evaluation of the variable-time MQ map of Algorithm 5. The difference is that on Line 4 we compute the monomial $t = x_i y_j + y_i x_j$ instead of $t = x_i x_j$. This only requires an additional look-up table multiplication and an additional XOR operation with respect to the evaluation of $\mathcal{P}$. For the private evaluation, we can group the multiplications in Equation 5 as

$$b(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} \Big[\Big(\sum_{j=i}^{n} a_{i,j} y_j \Big) x_i + \Big(\sum_{j=i}^{n} a_{i,j} x_j \Big) y_i \Big],$$

similarly to the approach adopted in Equation 4. We obtain the private evaluation by appropriately modifying the constant-time MQ map evaluation of Algorithm 5. The algorithm requires to bitslice and multiply m elements $2\big[\frac{n(n+1)}{2} + m\big]$ times. This is more efficient than trivially evaluating the polar form, which would require $3\big[\frac{n(n+1)}{2} + m\big]$ bitsliced multiplications of m elements.

4.4 New parameters for MAYO

In our analysis of implementation techniques for UOV-based schemes on 32-bit CPUs using the ARMv7-M architecture, we observed that bitsliced multiplications significantly enhance

Algorithm 6: Evaluation of Polar Form

Input: Macaulay matrix $\mathbf{M} \in \mathbb{F}^{m \times \binom{n+1}{2}}$ representing a MQ map $\mathcal{P}$
Input: $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^n$
Output: $\boldsymbol{z} = \mathcal{B}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{P}(\boldsymbol{x} + \boldsymbol{y}) - \mathcal{P}(\boldsymbol{x}) - \mathcal{P}(\boldsymbol{y}) \in \mathbb{F}^m$

EvalPolarConst($\mathbf{M}$, $\boldsymbol{x}$, $\boldsymbol{y}$):
1: $\boldsymbol{z} \leftarrow \mathbf{0}_m$
2: **for** $i \leftarrow 1, \ldots, n$ **do**
3: $\quad \text{acc} \leftarrow \mathbf{0}_m$
4: $\quad$ **for** $j \leftarrow i, \ldots, n$ **do**
5: $\quad\quad \left[\text{acc}_k\right]_{k\in[m]} \mathrel{+}\!\leftarrow x_j \cdot \left[\mathbf{M}_{i,j,k}\right]_{k\in[m]}$ ▷ bitsliced multiplication
6: $\quad \left[z_k\right]_{k\in[m]} \mathrel{+}\!\leftarrow y_i \cdot \left[\text{acc}_k\right]_{k\in[m]}$ ▷ bitsliced multiplication
7: $\quad \text{acc} \leftarrow \mathbf{0}_m$
8: $\quad$ **for** $j \leftarrow i, \ldots, n$ **do**
9: $\quad\quad \left[\text{acc}_k\right]_{k\in[m]} \mathrel{+}\!\leftarrow y_j \cdot \left[\mathbf{M}_{i,j,k}\right]_{k\in[m]}$ ▷ bitsliced multiplication
10: $\quad \left[z_k\right]_{k\in[m]} \mathrel{+}\!\leftarrow x_i \cdot \left[\text{acc}_k\right]_{k\in[m]}$ ▷ bitsliced multiplication
11: **return** $\boldsymbol{z}$

EvalPolarLeak($\mathbf{M}$, $\boldsymbol{x}$, $\boldsymbol{y}$):
1: $\text{acc} \leftarrow \mathbf{0}_{(q-1)\times m}$
2: **for** $i \leftarrow 1, \ldots, n$ **do**
3: $\quad$ **for** $j \leftarrow i, \ldots, n$ **do**
4: $\quad\quad t \leftarrow x_i y_j + x_j y_i \in \mathbb{F}$ ▷ 2 look-up table multiplications
5: $\quad\quad$ **if** $t = 0$ **then continue**
6: $\quad\quad$ **for** $k \leftarrow 1 \ldots, m$ **do**
7: $\quad\quad\quad \text{acc}_{t,k} \leftarrow \text{acc}_{t,k} + \mathbf{M}_{i,j,k}$
8: $\boldsymbol{z} \leftarrow \text{acc}_1 \in \mathbb{F}^m$
9: **for** $t \leftarrow \mathbb{F} \setminus \{0, 1\}$ **do**
10: $\quad \left[z_k\right]_{k\in[m]} \mathrel{+}\!\leftarrow t \cdot \left[\text{acc}_{t,k}\right]_{k\in[m]}$ ▷ bitsliced multiplication
11: **return** $\boldsymbol{z}$

the efficiency of both public and private components of the algorithm. In Section 4.1, we discussed the specific case where the base field is $\mathbb{F}_{16}$. In this scenario, each multiplication of m elements requires the application of both the bitslice and multiplication algorithms $\lceil m/32 \rceil$ times. Therefore, to achieve maximum performance from the bitslicing technique, the number of polynomials (i.e. m) appearing in the public map should ideally be a multiple of 32 or slightly less. The original parameters for MAYO do not satisfy this requirement, with m values of 67, 99, and 132 for security targets of 128, 192, and 256 bits, respectively.

When 8 or fewer field elements need to be bitsliced, they can be stored in a single register, streamlining the bitslicing and unbitslicing procedures. In Algorithm 10, the costs of these operations is reduced to 7 cycles and 4 cycles, respectively, compared to the original 28 cycles of Algorithm 7. However, the bitsliced multiplication instructions remain the same, irrespective of the number of elements involved. Consequently, multiplying 8 or fewer elements incurs the same computational cost as multiplying 32 elements once the elements are bitsliced. This inefficiency becomes particularly noticeable for the parameters of MAYO in Table 1, where we must separately multiply only 3 or 4 extra elements each time.

To address this issue, we propose new parameters for MAYO, ensuring that m is a multiple of 32 (see Table 2). By doing so, we can maximize the performance gains from the bitslicing technique, ultimately leading to a more efficient implementation on 32-bit ARMv7-M CPUs. The newly proposed parameters have been thoroughly analyzed for security and efficiency to ensure that they meet the required security levels while enhancing the overall performance of the scheme.

Whipping factor The whipping transformation of MAYO (Section 3.1) is largely determined by the parameter k. Smaller values of k lead to a smaller whipped map, resulting in faster signing and verification processes. On the other hand, larger values of k allow the size of the secret subspace to be reduced, thereby reducing the size of the keys. We attempted to achieve a smaller k while maintaining the correctness of the scheme through

Table 2: New parameters proposal for MAYO, with comparison between old and new parameters

Security level	Parameters						Pub. key (Bytes)	Priv. key (Bytes)	Sig. (Bytes)
	Set	n	m	o	k	q			
I	[Beu22b]	66	67	5	14	16	503	10,218	462
	This work	66	64	7	10	16	896	13,216	330
III	[Beu22b]	98	99	6	17	16	1,040	27,324	833
	This work	98	96	8	12	16	1,728	34,560	588
V	[Beu22b]	130	132	7	19	16	1,848	56,826	1,235
	This work	131	128	10	13	16	3,520	77,440	852

Table 3: Complexity ($\log_2$ of gates) of the attacks for the new parameter sets for MAYO

Security level	Direct	Reconc.	K-S	Inters.
I	145	146	249	287
III	211	212	371	426
V	277	275	493	565

the constraint $ko \geq m$, prioritizing compact public keys and signatures among the secure sets of parameters in our search.

Security analysis In defining new parameters, we adopted the same security targets originally identified in [Beu22b]. We used the estimates of the complexity of MAYO attacks provided in Section 3.3 to confirm that the new parameters align with their respective security targets. For the Direct Attack and the Reconciliation Attack, we used the "Multivariate Quadratic Estimator" of [BMSV22] to find the optimal algorithm and the respective complexity estimates for the proposed parameters. The SageMath code used in the analysis is available in the public repository.

Proposed paramters By setting the new values of m as 64, 96, and 128 respectively, we conducted an exhaustive search on the MAYO parameters to identify the optimal trade-off between efficiency and key/signature size. The newly proposed parameters are presented in Table 2, along with their security analysis in Table 3. These parameters, at the cost of slightly larger public keys, produce smaller k and marginally smaller signatures, ultimately resulting in a more efficient implementation of the MAYO scheme on 32-bit ARMv7-M CPUs.

Update of MAYO official parameters Recently, the MAYO team published a draft of the scheme specification document [BCC+23]. In the document, the authors propose new parameter sets for the purpose of optimized implementations. In particular, they choose the m parameter to be a multiple of 32. Although the authors' choice is not directly oriented toward optimization on ARM architecture, the underlying reasoning closely aligns with the discussion presented in this paper. Therefore, we are confident that the optimizations proposed in this work for the parameters of Table 2 can be easily adapted to the new official parameters proposed in [BCC+23].

Table 4: Performance evaluation of MAYO with first level parameters on ARMv7-M. Public and private keys are provided as Macaulay matrices in standard form without precomputation. The cycle count is reported as the average of 1000 iterations.

Parameters Set	Parameters					Cycles	
	n	m	o	k	q	Signature	Verification
[Beu22b]	66	67	5	14	16	114,378k	15,144k
This work	66	64	7	10	16	42,981k	5,703k
This work ($k = 14$)	66	64	7	14	16	85,682k	10,776k

5 Experimental results

The evaluation was carried out by applying the optimizations of the previous section to the reference implementation of MAYO [Beu22b]. Benchmarks were performed on parameters in Table 2 of the first security level. Implementing the key generation in the ARMv7-M instructions would not particularly benefit from the proposed optimization. Therefore, the analyses mainly focus on the signature and verification processes.

Data encoding Keys are generated as described in Algorithm 1 and are appropriately encoded. From the private key $(\mathbf{O}, \{\mathbf{P}_i^{(1)}, \mathbf{L}_i\}_{i\in[m]})$ and the public key $\{\mathbf{P}_i\}_{i\in[m]}$, we compute the Macaulay matrices $\mathbf{M}_{\mathcal{P}} \in \mathbb{F}^{m\times\binom{n+1}{2}}$, representing $\{\mathbf{P}_i\}_{i\in[m]}$, and $\mathbf{M}_{\mathcal{P}^{(1)}} \in \mathbb{F}^{m\times\binom{n-o+1}{2}}$, representing $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}$. Lastly, we pack the linear matrices $\mathbf{L}_i$ in the rows of $\mathbf{M}_L \in \mathbb{F}^{o(n-o)\times m}$.

Two consecutive $\mathbb{F}_{16}$ elements are packed in one byte where the first element occupies the least significant bits. As $\mathbf{M}_{\mathcal{P}}, \mathbf{M}_{\mathcal{P}^{(1)}}$ and $\mathbf{M}_L$ are stored in column-major order, if m is odd, we store each of their columns in $(m+1)/2$ bytes, where in the last byte we put the last element of the column and fill the remaining nibble with zeroes. Even though in this way the keys get slightly larger, different columns do not overlap, allowing faster access of the elements in the instruction sequences. The private matrix $\mathbf{O}$ must be in row-major order, so we apply the same reasoning if $n - o$ is odd.

Platform For the implementation, we used an ARM STM32H753 Nucleo-144 board, mounting an STM32H753ZIT6 microcontroller, which has a Cortex-M7 core that can run with a frequency of up to 480 MHz. The board has 2 Mbytes of Flash memory and 1 Mbyte of SRAM, which gives us enough space to store whole MAYO keys.

Message hashing The hash function H applied to messages in the signing process is the extendable output function SHAKE128. For its implementation we use th eXtended Keccak Code Package (XKCP) [1].

Results The implementation results are reported in Table 4. The verification procedure use variable-time algorithms and its running time heavily depends on the input signature. We remark that the signing process may require a restart if the system described in Section 4.2 does not have a solution; this happens with probability roughly 1/16, hence the signing runtime might not be constant. Therefore, we repeat both the signing and verification process for 1000 iterations and report the average running time. We expect that the choice of the whipping parameter k greatly impacts both the signing and the verification time, as the main operations are repeated $k + \binom{k+1}{2}$ times. To highlight the

[1] https://github.com/XKCP/XKCP

improvement deriving from the choiche of m being a multiple of 32, we include in the results the performance of the scheme with a variant of our new parameters sets, where we keep the original value $k = 14$.

The optimized implementation on the updated parameters reduces the latency of the signature process by 62.4% and that of the verification process by 62.3%. Without reducing the whipping parameter k to match the new parameters, we still obtain a reduction of 25.1% and 28.8% respectively.

6 Conclusions

We have explored the constant-time implementation of the MAYO multivariate digital signature scheme on ARMv7-M microcontrollers. The original parameters proposed for MAYO were not optimal for these embedded platforms. Therefore, we studied the security aspects and complexities of potential attacks and identified a new set of parameters that could improve the performance for the ARMv7-M platforms. This was achieved by finding a balance between efficiency and key sizes.

The primary idea was to maximize the efficiency of bitsliced multiplications, a technique that enables fast binary field arithmetic while enabling constant-time implementations. Our choice of parameters allowed for more efficient multiplications in groups of 32 field elements, resulting in signatures and verification runtimes that are 2.6 times faster compared to those of the original parameters. Although our parameters result in slightly larger public keys, they also produce shorter signatures.

Further improvements in performance could be achieved by using a simplified choice of parameter maps or alternative methods for finite field multiplications. However, a significant portion of the efficiency improvement is due to the reduction of the whipping parameter k, which affects the time complexity of the signing and verification processes.

Ultimately, the proposed parameters for the MAYO scheme demonstrate better performance on ARMv7-M platforms, offering an efficient, constant-time implementation for embedded systems while maintaining security against quantum threats.

Acknowledgements

This work was created with the co-financing of the European Union FSE-REACT-EU, PON Research and Innovation 2014-2020 DM1062/2021. The second author is a member of the INdAM Research Group GNSAGA. The core of this work is contained in the first author's M.Sc. thesis.

References

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the third round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2022. https://doi.org/10.6028/NIST.IR.8413-upd1.

[ABB+22] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. SPHINCS+ - Submission to the NIST post-quantum project. NIST PQC Standardization Process, version 1.2,

2022. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[ABD+21] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber - Algorithm Specifications And Supporting Documentation. NIST PQC Standardization Process, version 3.02, 2021. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[BCC+23] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias Kannwischer. MAYO - algorithm specifications. MAYO team, latest update: 28/02/2023, 2023. https://pqmayo.org/assets/specs/mayo.pdf, last accessed on 14/04/2023.

[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.

[BCS13] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 250–272. Springer, Heidelberg, August 2013.

[BDK+21] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium - Algorithm Specifications And Supporting Documentation. NIST PQC Standardization Process, version 3.1, 2021. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[Beu20] Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 183–211. Springer, Heidelberg, May 2020.

[Beu21] Ward Beullens. Improved cryptanalysis of UOV and Rainbow. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 348–373. Springer, Heidelberg, October 2021.

[Beu22a] Ward Beullens. Breaking rainbow takes a weekend on a laptop. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 464–479. Springer, Heidelberg, August 2022.

[Beu22b] Ward Beullens. MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 355–376. Springer, Heidelberg, September / October 2022.

[BFP09] Luk Bettale, Jean-Charles Faugere, and Ludovic Perret. Hybrid approach for solving multivariate systems over finite fields. *Journal of Mathematical Cryptology*, 3(3):177–197, 2009.

[BMSV22] Emanuele Bellini, Rusydi H. Makarim, Carlo Sanna, and Javier Verbel. An estimator for the hardness of the MQ problem. Cryptology ePrint Archive, Report 2022/708, 2022. https://eprint.iacr.org/2022/708.

[CFM+21] Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. GeMSS: A Great Multivariate Short Signature. NIST PQC Standardization Process, 2021. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 392–407. Springer, Heidelberg, May 2000.

[CKY21] Tung Chou, Matthias J. Kannwischer, and Bo-Yin Yang. Rainbow on cortex-M4. *IACR TCHES*, 2021(4):650–675, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9078.

[DCK+21] Jintai Ding, Ming-Shing Chen, Matthias Kannwischer, Jacques Patarin, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang. Rainbow. NIST PQC Standardization Process, 2021. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[Fau02] Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, page 75–83, New York, NY, USA, 2002. Association for Computing Machinery.

[FHK+20] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. NIST PQC Standardization Process, version 1.2, 2020. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar signature schemes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 206–222. Springer, Heidelberg, May 1999.

[KS98] Aviad Kipnis and Adi Shamir. Cryptanalysis of the Oil & Vinegar signature scheme. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 257–266. Springer, Heidelberg, August 1998.

[MHT13] Hiroyuki Miura, Yasufumi Hashimoto, and Tsuyoshi Takagi. Extended algorithm for solving underdefined multivariate quadratic equations. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 118–135. Springer, Heidelberg, June 2013.

[NIS23] NIST. Post-quantum cryptography: Digital signature schemes standardization project, 2023. https://csrc.nist.gov/projects/pqc-dig-sig/.

[Pat97] Jacques Patarin. The oil and vinegar algorithm for signatures. In *Dagstuhl workshop on cryptography, 1997*, 1997.

[PTBW11] Albrecht Petzoldt, Enrico Thomae, Stanislav Bulygin, and Christopher Wolf. Small public keys and fast verification for multivariate quadratic public key systems. Cryptology ePrint Archive, Report 2011/294, 2011. https://eprint.iacr.org/2011/294.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

[SS16] Peter Schwabe and Ko Stoffelen. All the AES you need on cortex-m3 and M4. In Roberto Avanzi and Howard M. Heys, editors, *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*, volume 10532 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 2016.

[TPD21] Chengdong Tao, Albrecht Petzoldt, and Jintai Ding. Efficient key recovery for all HFE signature variants. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 70–93, Virtual Event, August 2021. Springer, Heidelberg.

[TW12] Enrico Thomae and Christopher Wolf. Solving underdetermined systems of multivariate quadratic equations revisited. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 156–171. Springer, Heidelberg, May 2012.

A Instruction Sequences

In this appendix we show some key algorithms used in our implementation.

Please note that the `.w` suffix for some operations is due to the fact that ARMv7-M allows for instruction encoding in 16 and 32 bits, and these can be mixed. The result might lead to performance penalties due, for example, to having a 32-bit instruction split into two words, as originally observed in [SS16]. The `.w` suffix forces the encoding of instructions in 32 bit, thus avoiding misalignment.

Some of the following algorithms make use of the conditional execution of ARMv7-M instructions. These are the so called `it`-blocks, because they start with an `it` instruction and can comprise of up to 4 following instructions, depending on the number of `t` or `e` following the initial `it` instruction. The `it` instruction requires to have the flags already set, this is usually achieved through a `tst` instruction. Using conditional execution instructions is advantageous for achieving constant-time implementation, as it mitigates the risk of side-channel attacks by ensuring that the number of cycles will be consistent regardless of the secret input or condition flags.

Algorithm 7: Conversion of 32 $\mathbb{F}_{16}$ elements to bitsliced representation

Input: 32 $\mathbb{F}_{16}$ elements stored in normal representation in registers A0, A1, A2, A3
Output: Bitsliced elements in registers A0′, A1′, A2′, A3′

```
 1: and.w A0′, A0, #0x11111111
 2: and.w A3′, A1, #0x11111111
 3: orr.w A0′, A0′, A3′,lsl#1
 4: and.w A3′, A2, #0x11111111
 5: orr.w A0′, A0′, A3′,lsl#2
 6: and.w A3′, A3, #0x11111111
 7: orr.w A0′, A0′, A3′,lsl#3

 8: and.w A1′, A1, #0x22222222
 9: and.w A3′, A0, #0x22222222
10: orr.w A1′, A1′, A3′,lsr#1
11: and.w A3′, A2, #0x22222222
12: orr.w A1′, A1′, A3′,lsl#1
13: and.w A3′, A3, #0x22222222
14: orr.w A1′, A1′, A3′,lsl#2
15: and.w A2′, A2, #0x44444444
16: and.w A3′, A0, #0x44444444
17: orr.w A2′, A2′, A3′,lsr#2
18: and.w A3′, A1, #0x44444444
19: orr.w A2′, A2′, A3′,lsr#1
20: and.w A3′, A3, #0x44444444
21: orr.w A2′, A2′, A3′,lsl#1

22: and.w A3′, A3, #0x88888888
23: and.w A3, A0, #0x88888888
24: orr.w A3′, A3′, A3,lsr#3
25: and.w A3, A1, #0x88888888
26: orr.w A3′, A3′, A3,lsr#2
27: and.w A3, A2, #0x88888888
28: orr.w A3′, A3′, A3,lsr#1
```

Algorithm 8: Bitsliced multiplication of 32 $\mathbb{F}_{16}$ elements

Input: 32 $\mathbb{F}_{16}$ elements stored in bitsliced representation in registers A0, A1, A2, A3
Input: $\mathbb{F}_{16}$ factor element in least significant nibble of register B
Input: 32 $\mathbb{F}_{16}$ elements stored in bitsliced representation in accumulators C0, C1, C2, C3
Output: Bitsliced result of the multiplication in registers C0, C1, C2, C3

```
 1: eor.w T0, A0, A3
 2: eor.w T1, A2, A3
 3: eor.w T2, A1, A3

 4: tst.w B, #1
 5: itttt ne
 6: eorne.w C0, C0, A0
 7: eorne.w C1, C1, A1
 8: eorne.w C2, C2, A2
 9: eorne.w C3, C3, A3

10: tst.w B, #2
11: itttt ne
12: eorne.w C0, C0, A3
13: eorne.w C1, C1, T0
14: eorne.w C2, C2, A1
15: eorne.w C3, C3, A2

16: tst.w B, #4
17: itttt ne
18: eorne.w C0, C0, A2
19: eorne.w C1, C1, T1
20: eorne.w C2, C2, T0
21: eorne.w C3, C3, A1

22: tst.w B, #8
23: itttt ne
24: eorne.w C0, C0, A1
25: eorne.w C1, C1, T2
26: eorne.w C2, C2, T1
27: eorne.w C3, C3, T0
```

Algorithm 9: Constant-time inversion of an element in $\mathbb{F}_{16}$

Input: Non-zero element a of $\mathbb{F}_{16}$ in the least significant nibble of register A
Output: Inverse of a in register B

```
 1: movw T0, #0x4976
 2: lsr.w B, T0, A
 3: and.w B, #1

 4: movw T0, #0x53E8
 5: lsr.w T1, T0, A
 6: and.w T1, #1
 7: orr.w B, B, T1, lsl#1

 8: movw T0, #0x2DD8
 9: lsr.w T1, T0, A
10: and.w T1, #1
11: orr.w B, B, T1, lsl#2

12: movw T0, #0x953C
13: lsr.w T1, T0, A
14: and.w T1, #1
15: orr.w B, B, T1, lsl#3
```

Algorithm 10: Conversion of 8 $\mathbb{F}_{16}$ elements to and from bitsliced representation

Input: 8 $\mathbb{F}_{16}$ elements stored in normal representation in register `A`
Output: Bitsliced elements in registers `A0′`, `A1′`, `A2′`, `A3′`

```
1: and.w A0′, A, #0x11111111
2: and.w A1′, A, #0x22222222
3: lsr.w A1′, A1′, #1
4: and.w A2′, A, #0x44444444
5: lsr.w A2′, A2′, #2
6: and.w A3′, A, #0x88888888
7: lsr.w A3′, A3′, #3
```

Input: 8 $\mathbb{F}_{16}$ elements stored in bitsliced representation in registers `A0′`, `A1′`, `A2′`, `A3′`
Output: Normal elements in register `A`

```
1: and.w A, A0′, #0x11111111
2: orr.w A, A, A1′, lsl#1
3: orr.w A, A, A2′, lsl#2
4: orr.w A, A, A3′, lsl#3
```