Single-trace side-channel attacks on MAYO exploiting leaky modular multiplication

Sönke Jendral and Elena Dubrova

KTH Royal Institute of Technology, Stockholm, Sweden
{jendral,dubrova}@kth.se

Abstract. In response to the quantum threat, new post-quantum cryptographic algorithms will soon be deployed to replace existing public-key schemes. MAYO is a quantum-resistant digital signature scheme whose small keys and signatures make it suitable for widespread adoption, including on embedded platforms with limited security resources. This paper demonstrates two single-trace side-channel attacks on a MAYO implementation in ARM Cortex-M4 that recover a secret key with probabilities of 99.9% and 91.6%, respectively. Both attacks use deep learning-assisted power analysis exploiting information leakage during modular multiplication to reveal a vector in the oil space. This vector is then extended to a full secret key using algebraic techniques.

Keywords: Side-channel analysis · MAYO · Multivariate cryptography · Post-quantum digital signature · Key recovery attack

1 Introduction

The National Institute of Standards and Technology (NIST) recently announced the second-round candidates in its competition for additional post-quantum cryptographic (PQC) digital signature algorithms, which aims to find schemes based on different underlying mathematical problems and with different size and performance characteristics to already standardised algorithms [25,24]. Among the submissions selected by NIST for the second round is MAYO, a multivariate quadratic digital signature scheme designed to be existentially unforgeable under chosen message attacks (EUF-CMA) in the random oracle model [7]. EUF-CMA security means that an adversary with access to the public key and a signing oracle cannot generate a valid signature for a new message. The security of MAYO relies on the presumed hardness of the *Oil and Vinegar* (OV) problem and a variant of the *Multivariate Quadratic* (MQ) problem called the *multi-target whipped MQ* problem.

As in the previous PQC competition [27], it is important to assess the security of candidates' implementations to physical attacks. It is well understood that an implementation of a theoretically secure algorithm can in practice be broken by a side-channel or fault attack [33,34,16]. Identifying which components of an implementation are susceptible to physical attacks and how they can be modified to reduce information leakage or make fault injection more difficult provides guidance to product developers and helps to design effective countermeasures, which is especially important prior to the widespread deployment of any PQC algorithm.

Contributions: In this paper, we present two single-trace attacks on an implementation of MAYO using deep learning-assisted power analysis. Both attacks reveal a vector in the oil space, which can then be extended to a full secret key using known algebraic techniques [13,2,30]. To the best of our knowledge, this is the first work evaluating the resistance of a MAYO implementation to side-channel analysis. Previous work has focused on fault injection attacks on MAYO.

The presented attacks exploit information leakage during modular multiply-add operations. The first attack targets a matrix-matrix multiplication in the key expansion procedure, which is run as a part of the signing algorithm. The second attack targets a matrix-vector multiplication that forms the final step of the signature generation. Both attacks enable the recovery of the full secret key from a single power trace with probabilities of 99.9% and 91.6% respectively.

We also propose countermeasures against the presented attacks.

Organisation of the paper: The rest of this paper is organised as follows. Section 2 describes previous work. Section 3 provides background information on the MAYO algorithm. Section 4 outlines the experimental setup. Section 5 presents the side-channel attacks. Section 6 describes the trace preprocessing and neural network training. Section 7 presents the partial enumeration method. Section 8 describes the secret key recovery method. Section 9 summarises the experimental results. Section 10 discusses possible countermeasures against the attacks. Section 11 concludes the paper.

2 Previous work

This section gives an overview of previous attacks on multivariate signature schemes which make use of side-channel analysis or fault injection to recover a secret key. Specifically for MAYO, to the best of our knowledge, all previous attacks focused on fault injection [4,18], with no studies evaluating side-channel analysis to date.

Yi and Li [37] show several side-channel attacks on the enhanced Tame Transformation Signature scheme, that combine Differential Power Analysis (DPA) with fault injection (referred to as *fault analysis attacks*). The attacks first fix the values of several variables, before recovering parts of the secret key using DPA from the affine transformations or the central map. They simulate their attack using a hardware design and are able to recover the full secret key in a few hours from 2,000 traces. As countermeasures, they suggest masking and hiding, as well as several methods at the logic level.

Yi and Nie [38] use a similar approach on Unbalanced Oil and Vinegar (UOV) that again combines DPA on information leakage during the evaluation of polynomials, matrix-vector multiplications, and during vector additions with fault injection (also referred to as *fault analysis attacks*). They experimentally validate their attack on a SAKURA-G FPGA board and are able to recover the full secret key of UOV in a few hours from 4,000 traces.

Park et al. [28] present a Correlation Power Analysis (CPA) attack on Rainbow (under equivalent keys and random affine maps) and UOV (under equivalent keys). They use information leakage from a matrix-vector product to recover the secret affine map S and from it, the other affine map T, which together form the secret key. They experimentally validate their attack on Rainbow on an Atmel XMEGA128 processor and show that they are able to recover the full secret key from 30 traces. As countermeasures against the attack, they propose the use of masking and hiding techniques.

Pokorný et al. [32] also present a CPA attack on Rainbow that makes use of information leakage from a matrix-vector product and recovers the secret affine maps S and T. They experimentally validate their attack on an STM32F303 processor and are able to recover the subkeys of the secret key with probabilities between 0.75 and 0.95 from between 40 to 475 traces. They also propose a multiplicative masking countermeasure, that prevents the attack by performing multiplication using masked elements.

Aulbach et al. [2] describe a template-based side-channel attack on UOV that makes use of information leakage of the vinegar variables during multiplication with known constants. They are able to recover the vinegar values, and from these, an oil vector and finally the full oil space using a combination of the Kipnis-Shamir attack and the reconciliation attack. They experimentally validate their attack on an STM32F303RCT7 processor and show that they are able to recover the full secret key from a single trace with a probability greater than 97%. Their attack is similar to our first attack on MAYO except that (1) we use neural networks instead of templates for modelling information leakage, (2) MAYO uses the field $\mathbb{F}_{16}$ and UOV uses the field $\mathbb{F}_{256}$, and (3) we attack multiply-add operations instead of plain multiplications.

Sayari et al. [35] briefly discuss how side-channel analysis and fault injection attacks in their hardware implementation of MAYO could be performed in their discussion of countermeasures against physical attacks. For side-channel analysis attacks, they point out that an attacker might be able to recover secret values from vector-matrix multiplications, which are performed several times throughout the algorithm, using the method described in [2]. They further suggest the use of shuffling and parallelisation to increase the difficulty of performing such attacks.

There have also been a number of fault injection attacks on multivariate schemes [17,22,36,23,3,14], including MAYO [4,18]. Some of these attacks are capable of recovering the full secret key of the scheme from a single faulty signature with high probability.

Table 1. MAYO parameter sets from [9].

Parameter set	n	m	o	k	q	salt_len	digest_len	pk_seed_len	$f(z)$
MAYO_1	66	64	8	9	16	24	32	16	$f_{64}(z)$
MAYO_2	78	64	18	4	16	24	32	16	$f_{64}(z)$
MAYO_3	99	96	10	11	16	32	48	16	$f_{96}(z)$
MAYO_5	133	128	12	12	16	40	64	16	$f_{128}(z)$

3 Background

This section describes the key steps in the MAYO algorithm specification.

3.1 MAYO algorithm

MAYO is a multivariate quadratic digital signature scheme introduced by Beullens [7], based on the *Oil and Vinegar* (OV) signature scheme originally introduced by Patarin [29]. In the random oracle model, the security of MAYO is based on the assumed hardness of two problems, called the OV and *multi-target whipped Multivariate Quadratic* (MQ) problems. In OV schemes, the public key is a multivariate map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ of m n-variate quadratic polynomials $p_1(x), \ldots, p_m(x)$ over a finite field $\mathbb{F}_q$. The map features a trapdoor, which is a secret subspace $\mathbf{O}$ on which it vanishes. Using the trapdoor, it is possible to efficiently find a preimage $\mathbf{s}$ of a hash $\mathbf{t}$ such that $\mathcal{P}(\mathbf{s}) = \mathbf{t}$. The idea is to choose $\mathbf{s} = \mathbf{v} + \mathbf{o}$, where $\mathbf{v}$ is a random vector (sometimes called a vinegar vector) and $\mathbf{o}$ is in the so-called oil space $\mathbf{O}$. For such a construction, finding the preimage reduces to solving a system of linear equations (for more details, we refer to [7]). Without knowledge of the trapdoor, finding a preimage is assumed to be difficult, which is known as the MQ problem.

Distinguishing a map with a trapdoor from a fully random map is similarly assumed to be difficult and the corresponding problem is known as the OV problem. MAYO employs an optimisation to reduce the size of the public key by constructing a larger map $\mathcal{P}^*$ from a smaller map $\mathcal{P}$ before finding the preimage. Beullens refers to this process as "whipping up" the map and the resulting variant of the MQ problem that asks to find a preimage in $\mathcal{P}^*$ is therefore called the *multi-target whipped MQ* problem.

An overview over the possible sets of parameters for MAYO is given in Table 1. For further details we refer to the specification [9]. Note that these are the parameters from the first-round submission, as the parameters for the second-round submission have not yet been published at the time of writing. The implications of the tentative second-round parameters presented by Beullens [8] are briefly addressed in Section 10. We are focusing on MAYO_1 in this paper, though other variants can be approached similarly.

The main components of the MAYO scheme are the key generation algorithm, the signing algorithm and the verification algorithm.

Algorithm 1 MAYO.KeyGen() [7]

```
Output: Public key $pk$, secret key $sk$
 1: $\mathsf{seed}_{sk} \leftarrow \{0,1\}^\lambda$
 2: $(\mathsf{seed}_{pk}, \mathsf{O\_bytes}) \leftarrow \mathsf{SHAKE256}(\mathsf{seed}_{sk})$
 3: $\mathbf{O} \leftarrow \mathsf{Decode}(\mathsf{O\_bytes})$
 4: for $i$ from 1 to $m$ do
 5:     $\mathbf{P}_i^{(1)} \leftarrow \mathsf{Expand}(\mathsf{seed}_{pk} \parallel P1 \parallel i)$
 6:     $\mathbf{P}_i^{(2)} \leftarrow \mathsf{Expand}(\mathsf{seed}_{pk} \parallel P2 \parallel i)$
 7:     $\mathbf{P}_i^{(3)} \leftarrow \mathsf{Upper}(-\mathbf{OP}_i^{(1)}\mathbf{O}^T - \mathbf{OP}_i^{(2)})$
 8: return $(pk, sk) = ((\mathsf{seed}_{pk}, \{\mathbf{P}_i^{(3)}\}_{1 \le i \le m}), \mathsf{seed}_{sk})$
```

Algorithm 2 MAYO.Sign(sk, M) [7]

```
Input: Secret key sk, message $M$
Output: Signature $\sigma$
 1: $\mathsf{seed}_{sk} \leftarrow \mathsf{sk}$
 2: $(\mathsf{seed}_{pk}, \mathsf{O\_bytes}) \leftarrow \mathsf{SHAKE256}(\mathsf{seed}_{sk})$
 3: $\mathbf{O} \leftarrow \mathsf{Decode}(\mathsf{O\_bytes})$
 4: for $i$ from 1 to $m$ do
 5:     $\mathbf{P}_i^{(1)} \leftarrow \mathsf{Expand}(\mathsf{seed}_{pk} \parallel P1 \parallel i)$
 6:     $\mathbf{P}_i^{(2)} \leftarrow \mathsf{Expand}(\mathsf{seed}_{pk} \parallel P2 \parallel i)$
 7: $R \leftarrow \{0,1\}^r$                                  ▷ Deterministic variant: $R \leftarrow \{0\}^r$
 8: $\mathsf{salt} \leftarrow \mathsf{SHAKE256}(M \parallel R \parallel \mathsf{seed}_{sk})$
 9: $\mathbf{t} \leftarrow \mathsf{SHAKE256}(M \parallel \mathsf{salt})$
10: for $ctr$ from 0 to 255 do
11:     $\mathsf{V} \leftarrow \mathsf{SHAKE256}(M \parallel \mathsf{salt} \parallel \mathsf{seed}_{sk} \parallel ctr)$
12:     $\mathsf{v}_1, \ldots, \mathsf{v}_k \leftarrow \mathsf{Decode}(\mathsf{V})$
13:     $(\mathbf{A}, \mathbf{y}) \leftarrow \mathsf{BuildLinearSystem}(\{\mathsf{v}_1, \ldots, \mathsf{v}_k\}, \mathbf{O}, \mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \mathbf{t})$
14:     $\mathsf{x} \leftarrow \mathsf{SampleSolution}(\mathbf{A}, \mathbf{y})$        ▷ Try to find $Ax = y$ (i.e. $\mathcal{P}^*(\mathbf{s}) = \mathbf{t}$)
15:     if $\mathsf{x} \neq \perp$ then break
16: $\mathbf{s} \leftarrow \{\mathsf{v}_i + \mathbf{O}\mathsf{x}_i \parallel \mathsf{x}_i\}_{1 \le i \le k}$
17: return $\sigma = (\mathbf{s}, \mathsf{salt})$
```

Key generation (Algorithm 1) The key generation samples a secret random seed seed_{sk}, from which a public seed seed_{pk} and a secret matrix $\mathbf{O} \in \mathbb{F}_q^{o \times (n-o)}$ are derived. From the public seed, the sequences of m matrices $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(2)}$ of the multivariate quadratic map $\mathcal{P}$ are expanded pseudorandomly. This allows the public key to only contain the seed instead of the matrices, thereby reducing its size. Finally, the remaining sequence of m matrices $\mathbf{P}_i^{(3)}$ is chosen such that the map $\mathcal{P}$ vanishes on the oil space $\mathbf{O}$. The public key consists of the public seed seed_{pk} and the sequence of matrices $\mathbf{P}^{(3)}$. The secret key consists of the secret seed seed_{sk}.

Signing (Algorithm 2) The signing algorithm recomputes the oil space $\mathbf{O}$ and the sequences of matrices $\mathbf{P}^{(1)}$ and $\mathbf{P}^{(2)}$ from the secret key. It then computes the salt and the target value $\mathbf{t}$. From the message M, the salt, the secret seed

Algorithm 3 MAYO.Verify(pk, M, σ) [7]

```
Input: Public key $\mathsf{pk}$, message $M$, signature $\sigma$
Output: Boolean
1: $(\mathsf{seed}_{pk}, \mathbf{P}^{(3)}) \leftarrow pk$
2: for $i$ from 1 to $m$ do
3:     $\mathbf{P}_i^{(1)} \leftarrow \mathsf{Expand}(\mathsf{seed}_{pk} \parallel P1 \parallel i)$
4:     $\mathbf{P}_i^{(2)} \leftarrow \mathsf{Expand}(\mathsf{seed}_{pk} \parallel P2 \parallel i)$
5: $(\mathbf{s}, \mathsf{salt}) = \sigma$
6: $\mathbf{t} \leftarrow \mathsf{SHAKE256}(M \parallel \mathsf{salt})$
7: $\mathbf{t}' \leftarrow \mathsf{Evaluate}_{\mathbf{P}^{(1)},\mathbf{P}^{(2)},\mathbf{P}^{(3)}}(\mathbf{s})$        ▷ $\mathcal{P}^*(\mathbf{s}) = \mathbf{t}'$
8: return true if $\mathbf{t} = \mathbf{t}'$ else false
```

seed_{sk} and a counter value ctr, the vinegar values $\mathsf{v}_1, \ldots, \mathsf{v}_k$ are derived. Then, a system of linear equations is constructed and solved, which corresponds to finding $\mathbf{s}$ with $\mathcal{P}^*(\mathbf{s}) = \mathbf{t}$ for the multivariate quadratic map $\mathcal{P}^*$. If the matrix $\mathbf{A}$ does not have full rank, due to the choice of vinegar values, the system cannot be solved and the algorithm restarts with different vinegar values. The trapdoor in $\mathcal{P}^*$ allows the system to be solved efficiently, provided that the oil space $\mathbf{O}$ is known. The signature consists of the sequence $\mathbf{s}$ of vinegar-masked oil vectors $\mathsf{v}_1 + \mathbf{O}\mathsf{x}_1, \ldots, \mathsf{v}_k + \mathbf{O}\mathsf{x}_k$ concatenated with vectors $\mathsf{x}_1, \ldots, \mathsf{x}_k$, and the salt.

Verification (Algorithm 3) Given a signature $(\mathsf{s}, \mathsf{salt})$ for a message M and a public key pk, the verification algorithm simply needs to check if $\mathcal{P}^*(s) = \mathsf{Hash}(M \parallel \mathsf{salt})$. It does this by extracting the sequences of matrices $\mathbf{P}^{(1)}$, $\mathbf{P}^{(2)}$ and $\mathbf{P}^{(3)}$ from the public key. It also extracts the preimage $\mathbf{s}$, as well as the salt from the signature. It then derives the original target value $\mathbf{t}$ (by hashing the message using $\mathsf{SHAKE256}$) and evaluates the multivariate quadratic map $\mathcal{P}^*$ with the preimage $\mathbf{s}$ to compute the value $\mathbf{t}'$. If the resulting values $\mathbf{t}$ and $\mathbf{t}'$ match, the signature is valid.

4 Experimental setup

This section describes the equipment used in the experiments, as well as the target implementation of MAYO.

4.1 Equipment

The equipment used in our experiments is shown in Fig. 1. The CW308-STM32F4 target board contains an ARM Cortex-M4 STM32F415RGT6 processor running at a frequency of 24 MHz. It is mounted on a CW313 adapter board. Power traces are captured using a ChipWhisperer-Husky.

The trace capture is triggered via ARM CoreSight DWT watchpoints, thus avoiding any modification of the assembly code otherwise caused by inserting a trigger. Alternative trigger sources, such as communication with peripheral

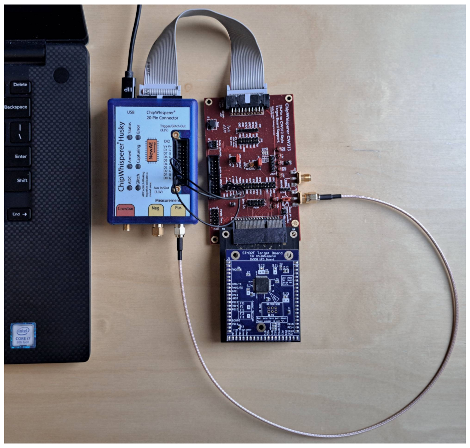

Fig. 1. Equipment used in the experiments: ChipWhisperer-Husky, CW313 adapter board (red) and CW308T-STM32F4 board (blue).

devices or similarity of the power consumption to reference waveforms, could be used by an attacker that does not have control over the target device.

4.2 Target implementation

In our experiments, we use the MAYO implementation by Beullens et al. [10]. Specifically, we use the most recent commit (`fe46236`) of the `main` branch, not the `nibbling-mayo` branch.

The first attack in this paper is specific to the bitsliced version in the `main` branch and does not work for the nibbled version that is expected to become the default in the second round [8]. The second attack is not specific to the bitsliced version, as the corresponding code is used in both versions.

It is worth pointing out that in the implementation by Gringiani et al. [15], exactly the same code is used for the two functions that we attack in this paper, though they are called from different procedures. In other words, our first attack on the key expansion in the implementation by Beullens et al. translates to an attack on the matrix-vector multiplication at the end of the signing algorithm in the implementation by Gringiani et al., and vice versa.

The implementation is compiled using `arm-none-eabi-gcc` with the highest optimization level `-O3` (recommended default).

5 Side-channel attacks

In this section, we describe the two side-channel attacks we conducted. Both attacks target multiplication operations involving the matrix $\mathbf{O}$ that forms the oil space. This is a $(n-o) \times o = 58 \times 8$ matrix with entries in $\mathbb{F}_{16}$. Due to known algebraic attacks, which we describe in Section 8, it is sufficient to recover a

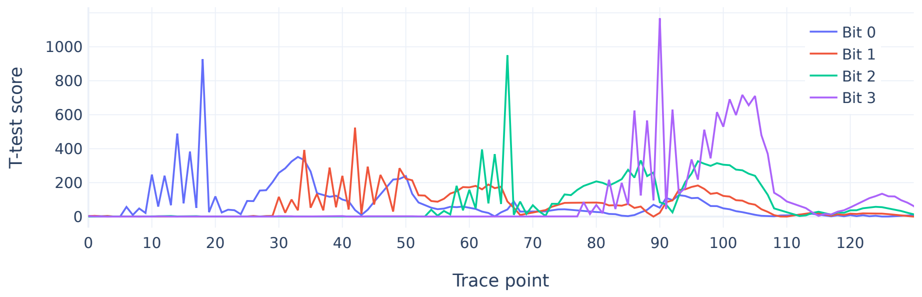

Fig. 2. T-test results for each bit of an entry of $\mathbf{O}$ during the processing by `gf16_madd_bitsliced`.

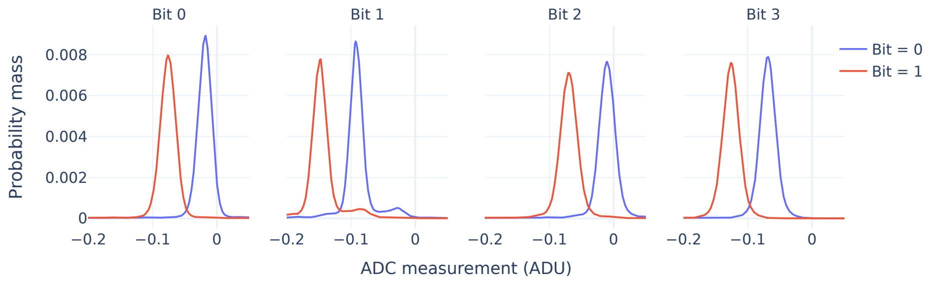

Fig. 3. Distributions of power consumption for each bit of an entry of $\mathbf{O}$ at the trace point of maximum t-test score during the processing by `gf16_madd_bitsliced`.

single vector in the span of the column vectors of $\mathbf{O}$. All other entries can then be derived from this vector. In our attacks, the vector we recover is exactly one of the column vectors of $\mathbf{O}$ (as opposed to a linear combination of them), thus the goal of the attacker is to acquire the 58 4-bit entries of this vector through side-channel analysis.

5.1 First attack: `gf16_madd_bitsliced`

The first attack exploits information leakage in the `gf16_madd_bitsliced` procedure, which is part of the matrix multiplication to compute a sequence of matrices $(\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)T})\mathbf{O}$ during the key expansion algorithm. In the implementation by Beullens et al. [10], the key expansion is executed prior to the generation of the signature on every invocation of the signing algorithm.

The assembly code for the `gf16_madd_bitsliced` procedure is given in Listing 1.1. The leakage is caused by the conditional execution of the highlighted blocks of instructions. Each block begins with a `tst` instruction, which tests the value of a specific bit of an entry of $\mathbf{O}$ and, depending on the value, either exe-

```
eor.w r11, r7, r10
eor.w r12, r9, r10
eor.w r14, r8, r9

tst.w r0, #1
nop.n
itttt ne
eorne.w r3, r3, r7
eorne.w r4, r4, r8
eorne.w r5, r5, r9
eorne.w r6, r6, r10
tst.w r0, #2
nop.n
itttt ne
eorne.w r3, r3, r10
eorne.w r4, r4, r11
eorne.w r5, r5, r8
eorne.w r6, r6, r9
tst.w r0, #4
nop.n
itttt ne
eorne.w r3, r3, r9
eorne.w r4, r4, r12
eorne.w r5, r5, r11
eorne.w r6, r6, r8
tst.w r0, #8
nop.n
itttt ne
eorne.w r3, r3, r8
eorne.w r4, r4, r14
eorne.w r5, r5, r12
eorne.w r6, r6, r11
```

Listing 1.1. The assembly code of the `gf16_madd_bitsliced` procedure. The blocks of instructions that are conditionally executed according to on the values of the bits are highlighted in color.

cutes or skips the following instructions with an `it` pseudoinstruction. As noted by Chou et al. [12] for their implementation of Rainbow, from which this code is derived [10], the ARM v7-m architecture reference manual [1, Section A4.1.2] states that conditional instructions, whose condition is not fulfilled, behave as `NOP`s. As both the `eor` instruction and the `NOP` instruction are single-cycle instructions, the procedure executes in constant time, regardless of the value of the bits (i.e. there is no timing leakage). The lack of timing leakage could suggest that such an implementation is secure against side-channel attacks, as is claimed in e.g. [15]. However, our findings show that this interpretation is not correct in the context of power side-channel attacks. The differences in power consumption between the instructions being executed (i.e. the bit having value "1") or skipped (i.e. the bit having value "0") are significant (see Figs. 2 and 3) and can be used to recover the value of the entry.

A property of the matrix multiplication implementation useful to attackers is that the individual entries of $\mathbf{O}$ are processed by `gf16_madd_bitsliced` $\frac{m}{32} \cdot (n - o - 1) = 114$ times. The specifics of how this multiplication is carried out are not relevant here, but essentially, the implementation can compute 32 of the matrices in parallel, hence the entire procedure must be performed twice. Further, because the matrices $\mathbf{P}_i^{(1)}$ are upper-triangular and $\mathbb{F}_{16}$ has characteristic 2, each entry of $\mathbf{O}$ is involved in 57 computations because the entries along the diagonal are 0. An attacker can thus not only choose which of the $o = 8$ column vectors of $\mathbf{O}$ to recover, but also combine information from any of the 114 computations. In our attack, we recover the second column vector of $\mathbf{O}$ (which we found to leak slightly stronger than the first column vector) from the first set of the 114 computations. As the entries along the diagonals are 0, this set of computations did not include any leakage for the first entry of the column vector, which is instead simply enumerated.

5.2 Second attack: `mul_f` and `decode`

The second attack exploits leakage in the `mul_f` procedure called from the `mat_mul` procedure, which is responsible for computing the oil vectors $\mathbf{O}\mathsf{x}_i$ at the end of the signing algorithm.

The C and assembly code for the `mul_f` procedure is shown in Listings 1.2 and 1.3. The procedure performs multiplication of an entry of $\mathbf{O}$ (`a` in the C code) with an entry of a vector x_i (`b` in the C code) modulo the polynomial $x^4 + x + 1$. It does this by performing several multiplications shown in lines 3 to 6 of Listing 1.2 before combining the results in lines 9 and 10. The t-test results in Fig. 4 and the distributions of power consumption at the trace point of maximum t-test score in Fig. 5 show that three of the four bits of the entry of $\mathbf{O}$ leak information.

Intriguingly, one of the bits, bit 2 (which corresponds to the operation `(a & 4)*b` in line 5 of the C code), does not appear to leak any information. The assembly code in Listing 1.3 provides some clues about this phenomenon. We see that the operation `(a & 1)*b` in line 3 in the C code is realised as a signed bitfield extraction of the lowest bit in line 4 of the assembly code, followed by a

```c
unsigned char mul_f(unsigned char a, unsigned char b) {
    unsigned char p;
    p  = (a & 1)*b;
    p ^= (a & 2)*b;
    p ^= (a & 4)*b;
    p ^= (a & 8)*b;

    // reduce mod x^4 + x + 1
    unsigned char top_p = p & 0xf0;
    unsigned char out = (p ^ (top_p >> 4) ^ (top_p >> 3)) & 0x0f;
    return out;
}
```

Listing 1.2. C code of the `mul_f` procedure. The multiplications that depend on the values of the bits are highlighted in color.

```
...
ldrb.w  r4, [r6, #-8]    # Entry of O
ldr     r7, [sp, #28]    # Entry of xi
sbfx    ip, r4, #0, #1
and.w   r3, r4, #2
smulbb  r3, r3, r7
and.w   ip, ip, r7
eor.w   ip, ip, r3
ldr     r7, [sp, #28]
and.w   r5, r4, #4
smulbb  r5, r5, r7
and.w   r4, r4, #8
smulbb  r4, r4, r7
eor.w   r5, ip, r5
eors    r5, r4
...
```

Listing 1.3. Simplified excerpt of the assembly code of the `mul_f` procedure. The multiplications that depend on the values of the bits are highlighted in color.

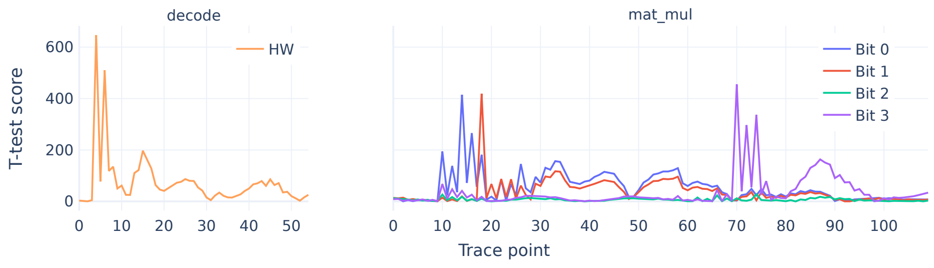

Fig. 4. T-test results for each bit of an entry of **O** and its Hamming weight during the processing by `decode` and `mul_f`.

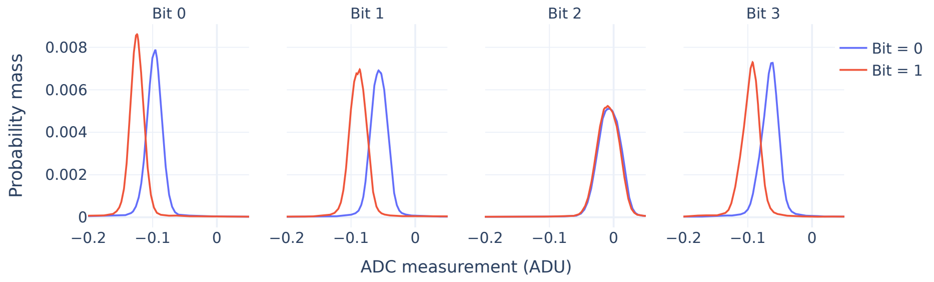

Fig. 5. Distributions of power consumption for each bit of an entry of **O** at the trace point of maximum t-test score during the processing by `mul_f`.

```c
void decode(unsigned char *m, unsigned char *mdec, int mdeclen) {
    int i;
    for (i = 0; i < mdeclen / 2; ++i) {
        *mdec++ = m[i] & 0xf;
        *mdec++ = m[i] >> 4;
    }

    if (mdeclen % 2 == 1) {
        *mdec++ = m[i] & 0x0f;
    }
}
```

Listing 1.4. C code of the `decode` procedure. The operations that store individual entries are highlighted in color.

bitwise AND in line 7. The signed bitfield extraction sign-extends its extracted value, which is simply the lowest bit of `a`. In our case, the `ip` register thus contains either the value `0x0` or the value `0xffffffff` and the bitwise AND in line 7 results in the value `0` or the value `b`. It is not clear why the compiler used a signed bitfield extraction and a bitwise AND instead of a bitwise AND followed by a signed multiplication, as is the case for the other bits. However, the difference in Hamming weight between the two possible values leaks information about the value of the lowest bit of `a`. The other operations from the C code follow a different pattern and are realised by a bitwise AND followed by a signed multiplication. A reasonable assumption would be that the signed multiplication of the corresponding bit of `a` with `b` would leak information. Indeed, we observe this behaviour for bits 1 (corresponding to `(a & 2)*b`) and 3 (corresponding to `(a & 8)*b`) in Figs. 4 and 5. However, the same is not true for bit 2, for which we observe no exploitable leakage.

In fact, even in an isolated case where we utilise the exact same sequence of instructions and registers and only change the immediate value in the bitwise AND, we observe leakage for the values 2 and 8, but no leakage for the value 4. We further observe no leakage for values 1, 16 or 64, but do observe leakage for the values 32 and 128. Our best hypothesis is that the ARM Cortex-M4 contains an optimisation for odd powers of 2, likely by performing bitshifts in these cases and that these bitshifts are causing the leakage we observe for bits 1 and 3 (while bit 0 leaks because it uses signed bitfield extraction and bit 2 does not leak, because it uses multiplication by an even power of 2). We are not able to definitively confirm that this optimisation exists, nor are we able to explain why this optimisation does not apply to even powers of 2, but in Section 10, we provide an alternative implementation of this part of the `mul_f` procedure that reduces leakage by replacing multiplications by 2 and 8 with multiplications by 64 and 16.

In the context of our attack, the lack of leakage for bit 2 means that we need to either enumerate its value for each of the 58 entries, or obtain additional information from a different leakage point. We chose to include leakage of the Hamming weight of the entry during the `decode` procedure, which is called during both the key expansion and signature generation. The C code of the `decode` procedure is shown in Listing 1.4. The procedure takes a sequence of bytes and splits each byte into two 4-bit entries. Storing these 4-bit entries (see lines 4, 5 and 9 of Listing 1.4) leaks their Hamming weight. Fig. 4 shows this leakage via t-test results obtained by partitioning the traces into two sets $\mathbf{T}_A$ and $\mathbf{T}_B$, such that $\mathbf{T}_a$ contains the traces where the entry being processed has Hamming weight < 2 and $\mathbf{T}_B$ contains the traces where the entry being processed has Hamming weight > 2. By combining knowledge about three of the four bits with knowledge of the Hamming weight of all four bits, we are able to recover the missing bit without enumeration.

As in the first attack, the operations targeted in this attack are also performed multiple times during the generation of a single signature, which can be used to increase the success probability of the attack. Specifically, because in

Table 2. Neural network architectures used for oil vector recovery.

Layer type	Output shape Attack 1	Attack 2 `mul_f`	`decode`
Batch Normalization 1	140	990	110
ReLU	140	990	110
Dense 1	128	128	128
Batch Normalization 2	128	128	128
ReLU	128	128	128
Dense 2	64	64	64
Batch Normalization 3	64	64	64
ReLU	64	64	64
Dense 3	32	32	32
Batch Normalization 4	32	32	32
ReLU	32	32	32
Softmax	16	8	5

the implementation of Beullens et al. [10] the key expansion is performed during every signature generation and both the key expansion and the signature generation procedures perform the `decode` operation, we are able to combine information from both points. Similarly, because all $k = 9$ oil vectors $\mathbf{O}\mathsf{x}_i$ require multiplication with $\mathbf{O}$, we are able to combine information from all 9 points.

6 Trace preprocessing and neural network training

In this section, we describe the trace preprocessing and neural network training process for our attacks. We use a profiled deep learning-assisted attack method based on the Hamming weight leakage model.

6.1 First attack: `gf16_madd_bitsliced`

For profiling, we use a dataset containing 570,000 trace segments, obtained by capturing $t = 10,000$ traces $\{T_1, \ldots, T_t\}$ for known matrices $\{\mathbf{O}_1, \ldots, \mathbf{O}_t\}$. We apply the cut-and-join technique of [26] to divide each trace T_i, $i \in \{1, \ldots, t\}$, into $n - o - 1 = 57$ segments $T_i[j]$, $j \in \{1, \ldots, 57\}$, (recall from Section 5.1 that the first entry is not leaked in the section we selected for capture, hence no segment for it is created). Each individual segment covers the multiplication of one entry $\mathbf{O}_i[j, 2]$ of the second column vector of $\mathbf{O}_i$ and the segments are labeled with the corresponding values.

We use a multilayer perceptron (MLP) neural network with the architecture shown in Table 2. The network is of type $\mathcal{N} : \mathbb{R}^{140} \to \{0, \ldots, 15\}$, where 140 is the number of sample points in each segment. The network maps each segment $T_i[j]$ into a score vector $S_{i,j} = \mathcal{N}(T_i[j])$, such that $S_{i,j}[c]$ represents the probability that $\mathbf{O}_i[j,2]$ takes value $c \in \{0,\dots 15\}$:

$$S_{i,j}[c] = \Pr[\mathbf{O}_i[j,2] = c].$$

During training, we use the Nadam optimiser with a learning rate of 0.01 and a numerical stability constant $\epsilon = 10^{-8}$. We train for a maximum of 100 epochs using early stopping with a patience of 15 and a batch size of 1024, and use 70% of the data for training and 30% for validation.

6.2 Second attack: `mul_f` and `decode`

For profiling, we use two datasets each containing $580,000$ trace segments, obtained by capturing $t = 10,000$ traces for known matrices $\{\mathbf{O}_1,\dots,\mathbf{O}_t\}$. Due to limitations in the maximum capture size of our equipment, these traces are obtained from four separate sets of captures with the same matrices $\{\mathbf{O}_1,\dots,\mathbf{O}_t\}$. Three of the four sets each contain the computation of three of the $k = 9$ oil vectors $\mathbf{O}_i \mathsf{x}_{i,l}$ and the fourth set contains both `decode` operations involving $\mathbf{O}_i$, the first during the key expansion and the second during the signature generation. With different equipment that supports larger capture sizes, it would be possible to capture all of these sets in a single capture. Thus, going forward, we consider the set of traces $\{T_1,\dots,T_t\}$ where each trace T_i, $i \in \{1,\dots,t\}$, is the concatenation of the ith trace of each of the four sets.

We again apply the cut-and-join technique of [26], however, this time we select the $n - o = 58$ segments in a slightly different manner. To create the first dataset $\{T_1^{\text{mul\_f}},\dots,T_t^{\text{mul\_f}}\}$, we divide each trace T_i into $(n-o)\cdot k = 58 \cdot 9$ segments covering the multiplication of one entry $\mathbf{O}_i[j,1]$, $j \in \{1,\dots,58\}$, of the first column vector of $\mathbf{O}_i$ with one entry of each of the k vectors $\mathsf{x}_{i,l}[j]$, $l \in \{1,\dots,k\}$. We then concatenate the nine segments where the entry $\mathbf{O}_i[j,1]$ is the same. Hence, each trace $T_i^{\text{mul\_f}}$ consists of 58 segments, where each segment j covers 9 multiplications involving the entry $\mathbf{O}_i[j,1]$.

To create the second dataset $\{T_1^{\text{decode}},\dots,T_t^{\text{decode}}\}$, we similarly divide each trace T_i into $58 \cdot 2$ segments covering the decoding of one entry $\mathbf{O}_i[j,1]$, $j \in \{1,\dots,58\}$, of the first column vector of $\mathbf{O}_i$ during the key expansion or during the signature generation. We then concatenate both segments where the entry $\mathbf{O}_i[j,1]$ is the same. Thus, each trace T_i^{decode} consists of 58 segments, were each segment j covers both `decode` operations involving the entry $\mathbf{O}_i[j,1]$.

We again use MLP networks with the architectures shown in Table 2. The networks are of types $\mathcal{N}_{\text{mul\_f}} : \mathbb{R}^{990} \to \{0,\dots,7\}$ and $\mathcal{N}_{\text{decode}} : \mathbb{R}^{110} \to \{0,\dots,4\}$. The first network $\mathcal{N}_{\text{mul\_f}}$ maps each trace segment $T_i^{\text{mul\_f}}[j]$ to a score vector $S_{\text{mul\_f},i,j}$ such that its cth entry represents the probability that bits 0, 1, and 3 of $\mathbf{O}_i[j,2]$ together take value c (recall from Section 5.2 that the value of bit 2 cannot be recovered in this way). The second network $\mathcal{N}_{\text{decode}}$ instead maps each trace segment $T_i^{\text{decode}}[j]$ to a score vector $S_{\text{decode},i,j}$ such that its cth entry represents the probability that the Hamming weight of $\mathbf{O}_i[j,2]$ takes value c.

We again use the Nadam optimiser with a learning rate of 0.01 and $\epsilon = 10^{-8}$, and we also train for a maximum of 100 epochs with early stopping with a patience of 15 and a batch size of 1024, and we use 70% of the data for training and 30% for validation.

7 Partial enumeration method

As mentioned in Section 5.2, certain entries cannot be recovered using side-channel analysis, because they are not processed by the device. In practice, we also find that neural network models are not able to recover entries with perfect accuracy. In this section, we describe enumeration methods for both of the presented attacks that are able to increase the success probability of the presented attacks.

Following [19], we use the maximum predicted class probabilities to guide the selection of entries (or bits) to enumerate.

For the first attack on the `gf16_madd_bitsliced` procedure, we enumerate all four bits of an entry j if the maximum predicted class probability $\max(S_{i,j}) < \alpha = 0.99999$. The value α was chosen empirically to maximise the number of entries which are enumerated without exceeding a maximum enumeration cost of 2^{32}.

For the second attack on the `mul_f` and `decode` procedures, we enumerate all four bits of an entry j if the maximum predicted class probabilities for the value of bits 0, 1, and 3, $\max(S_{\text{mul\_f},i,j})$, and the Hamming weight of the entry, $\max(S_{\text{decode},i,j})$, are both lower than $\alpha = 0.99999$. If $\max(S_{\text{decode},i,j}) < \alpha$, but $\max(S_{\text{mul\_f},i,j}) \geq \alpha$, we enumerate only the value of bit 2. Lastly, if $\max(S_{\text{decode},i,j}) \geq \alpha$ and $\max(S_{\text{mul\_f},i,j}) \geq \alpha$, we compute the value of bit 2 from the difference between the predicted Hamming weight and the Hamming weight of the prediction for bits 0, 1, and 3, without enumeration, unless the difference between the Hamming weights is larger than 1, in which case we again enumerate all four bits. As before, α was selected empirically to maximise the number of entries which are enumerated without exceeding a maximum enumeration cost of 2^{32}.

8 Secret key recovery method

Both attacks presented in Section 5 allow the attacker to recover a single vector in the oil space $\mathbf{O}$. In this section, we describe known methods that enable the recovery of the full secret key of OV schemes using knowledge of a single oil vector.

The reconciliation attack [13] allows finding additional vectors in the oil space by solving systems of linear equations. The Kipnis-Shamir attack [21,20] instead finds oil vectors by identifying eigenvectors of specifically chosen matrices. Aulbach et al. combined both the reconciliation attack and the Kipnis-Shamir attack in their attack on UOV [2] to recover the full oil space from a single oil vector. Beullens' intersection attack [6,7] also combines ideas from the reconciliation attack and the Kipnis-Shamir attack to recover two oil vectors simultaneously, which can be used in the reconciliation attack to recover the full oil space.

Pébereau [30] improved upon these approaches with an efficient algorithm that recovers the full oil space from a single vector, using the kernels of the matrices representing the public key. The results are applicable to UOV and UOV-like schemes, like MAYO.

In this paper, we use the approach from the reconciliation attack [13] to recover the secret key from the vector $\mathbf{o}_1$ in the oil space. The idea behind the attack, as described in [7], is to use the vector $\mathbf{o}_1$ to create a system of equations

$$\begin{cases} \mathcal{P}^*(\mathbf{o}_i) = 0 \\ \mathcal{P}^*(\mathbf{o}_1, \mathbf{o}_i) = 0 \end{cases}$$

and then solve this system (under o additional affine constrains) to find vectors $\mathbf{o}_i$ (with $i \in \{2, \dots, m\}$) to form a basis. In this way, the original oil space $\mathbf{O}$ is recovered. In practice, the necessary computations can be performed in less than a second.

9 Experimental results

This section describes the results of our side-channel attacks.

We train neural networks as described in Section 6 to perform recovery of an oil vector, additionally making use of the enumeration method described in Section 7. The neural networks are trained and tested on a PC with an AMD Ryzen 7 Pro 5850U running at 1.9 GHz and 32GB RAM. The stated probabilities are empirical probabilities (mean over 1000 secret keys selected at random) for recovering an oil vector from a single trace. Recall that, due to the techniques mentioned in Section 8, knowledge of a single oil vector is sufficient to recover the full secret key.

In the first attack on `gf16_madd_bitsliced`, we can recover an oil vector in 99.9% of 1000 attempts from a single trace, with an average enumeration of 8.30 bits and a maximum enumeration of 24 bits. When restricting enumeration to only the 4 bits that cannot be recovered from side-channel leakage, the attack still succeeds with the probability of 98.5%.

In the second attack on `mul_f` and `decode`, we can recover an oil vector in 91.6% of 1000 attempts from a single trace, with an average enumeration of 23.18 bits and a maximum enumeration of 32 bits. Without enumeration, the attack succeeds with the probability of 50.4%.

10 Countermeasures

This section discusses possible countermeasures against the presented side-channel attacks. We first describe countermeasures that are applicable to both attacks, and then suggest countermeasures specific to the second attack against the `mul_f` and `decode` procedures.

10.1 Countermeasures common to both attacks

A simple approach to increase the difficulty of the presented attacks is to increase the number of entries that need to be recovered, i.e. to increase the value $n - o$. As the recovery probability for a full oil vector is approximately exponential in the number of entries, even a small increase in the number of entries can cause a significant decrease in the recovery probability. The tentative second-round parameters presented by Beullens [8] suggest that the second-round submission for MAYO will likely feature an increase in $n - o$ (albeit with a different underlying motivation). Our rough estimates, based on the data captured in this paper, however, suggest that the proposed increase is not sufficient to make the presented attacks infeasible. We estimate that the probability for attack 1 decreases to 72.9% and the probability for attack 2 decreases to 82.23% if the number of entries increases from 58 to 78.

A second possible approach is to store the expanded secret key, instead of recomputing it each time during the signature generation. The choice to recompute the expanded key was likely made to reduce the overall memory consumption of the algorithm, while real-world implementations will presumably opt to store the expanded key instead (not least of all because the key expansion accounts for a significant portion of the runtime of the signing algorithm). If the key expansion is only performed once (for example after key generation), the first presented attack can only be conducted by attackers that are able to observe this process, which is a significantly smaller window of opportunity. For the second presented attack, reducing the number of times that the key expansion is performed would eliminate one of the two sources of leakage for the Hamming weight during the `decode` operation, but the other source of leakage for the Hamming weight and the leakage of three of the bits would be unaffected, thus the attack would likely still be applicable.

A third approach would be to ensure that computations involving entries of $\mathbf{O}$ are performed in parallel, whenever possible, as suggested in [35]. Both of the attacks presented in this paper make use of the fact that entries of $\mathbf{O}$ are, at some point, processed one-by-one. The nibbled implementation presented by Beullens et al. [10] makes progress towards this goal by replacing bitsliced matrix multiplications with table lookups. Notably, however, due to concerns about side-channel leakage [10], these table lookups cannot be used when the matrices being multiplied are not public, as is the case for $\mathbf{O}$. This is the reason why the second attack is applicable to both the bitsliced and nibbled version, as the latter does not replace the vulnerable parts with a different implementation. It may be the case that the matrix-vector multiplications $\mathbf{O}\mathsf{x}_i$ can be realised differently from matrix-matrix multiplications using table lookups or using a different technique that processes multiple entries simultaneously.

Lastly, known generic countermeasures, such as masking and shuffling [11], can also be used to make the presented attacks more difficult. The idea behind masking, as described in [11] is to split a value into multiple shares, perform computation on the shares and then combine the results. In that way, leakage during the computation only reveals information about the shares, not the value.

```
1  ...
2  ldrb.w  r4, [r6, #-8]      # Entry $a = a_3a_2a_1a_0$ of $\mathbf{O}$
3  ldr     r7, [sp, #28]      # Entry $b = b_3b_2b_1b_0$ of $\times_i$
4  and.w   ip, r4, #1
5  smulbb  ip, ip, r5
6  and.w   r3, r4, #4
7  smulbb  r3, r3, r5
8  eor.w   ip, ip, r3
9  eor.w   r4, r4, r7, lsl #4 # Mask HW of $a$ with $b$
10 rbit    r4, r4             # Convert $b_3b_2b_1b_0a_3a_2a_1a_0$
11 rev     r4, r4             # into $a_0a_1a_2a_3b_0b_1b_2b_3$
12 and.w   r5, r4, #16
13 smulbb  r5, r5, r7
14 eor.w   r5, r5, ip, lsl #1
15 and.w   r4, r4, #64
16 smulbb  r4, r4, r7
17 eor.w   r5, r4, r5, lsl #4
18 mov.w   r5, r5, lsr #5
19 ...
```

Listing 1.5. Excerpt of the modified assembly code of the `mul_f` procedure to reduce leakage.

Shuffling, on the other hand, randomises the order of execution, thus even if an attacker can successfully recover correct values, they do not know the order of the values. In practice, a number of successful side-channel attacks have been conducted on masked and shuffled implementations [31,26,5,19], thus care must be taken when implementing these countermeasures.

10.2 Countermeasures specific to the second attack

As mentioned in Section 5.2, it is possible to reduce the information leakage in the `mul_f` procedure by avoiding multiplications with odd powers of 2. Listing 1.5 shows a possible approach. Recall that the behaviour being realised here is that of lines 3 to 6 of the C code in Listing 1.2, which may be easier to understand than the assembly code.

The idea is to move the bits of the entry a of $\mathbf{O}$ that originally were in positions corresponding to odd powers of 2 so that they are in positions that correspond to even powers of 2, extract the bits in these positions, perform the - now not leaky - multiplications and shift the result so that the procedure has the same functionality.

A central challenge is to move the problematic bits into the correct positions without resorting to bitshifts or multiplications, as these would produce leakage. Our approach is to reverse the order of bits of the register containing a using a `rbit` instruction (see line 10 of Listing 1.5) and then reverse the byte order using a `rev` instruction (line 11). The effect of this is that bit 1, which originally corresponded to the leaky multiplication by 2^1, now corresponds to the non-leaky multiplication by 2^6 and bit 3, which originally corresponded to the leaky multiplication by 2^3, now corresponds to the non-leaky multiplication by 2^4.

However, we found that the `rbit` and `rev` instructions still leak information about the Hamming weight of a, which is indeed also the case for the initial load of a in line 2. For the `rbit` and `rev` instructions, we can mitigate the leakage by masking the Hamming weight with b (i.e. by setting the upper 4 bits of a to b), which makes the Hamming weight dependent on both a and b. This is not a useful approach in general, because the value of b is part of the signature and thus known to an attacker. A better approach is to store entries of $\mathbf{O}$ in such a way that their upper 4 bits always contain a random value, for example by using the output of $\mathsf{SHAKE256}$ directly, instead of applying the `decode` operation first (which would coincidentally also remove the `decode` operations as another source of leakage for the Hamming weight). We did not implement this approach, as it would be incompatible with the current specification.

After performing the multiplications, the results must be shifted and accumulated together, so that the modified procedure has the same functionality as the original procedure. For bit 3, where we performed multiplication by 2^4 instead of 2^3, we need to shift the result right by one bit before XORing it with the results for bit 0 and 2. Similarly, for bit 1, where we performed multiplication by 2^6 instead of 2^1, we need to shift the result right by five bits before XORing it with the results for bit 0, 2 and 3. However, we found that it is preferable to instead shift the accumulated result from bit 0 and 2 to the left by one bit, perform the XOR with the result from bit 3, shift the combined value to the left by four more bits, perform the XOR with the result from bit 1, and finally shift the combined value to the right by five bits. These steps are functionally equivalent, but each of the XOR operations now leak information about the Hamming weight of the combined value, instead of the individual results, which is more difficult for an attacker to exploit.

Note that we do not claim that this implementation is entirely resistant to side-channel attacks. Importantly, when we claim that multiplication by even powers of 2 does not leak information, we mean that the power consumption of the device is not fully dictated by a. It might be possible to use knowledge of b to distinguish the values of a, even if only multiplications by even powers of 2 are used, though this might be counteracted by masking the value of b. Additionally, because the value of $\mathbf{O}$ generally remains the same for multiple signatures, an attacker might be able to combine information from several traces, thereby amplifying small leakages, and potentially succeed with recovering the entries regardless. Nonetheless, the idea of replacing multiplications by odd powers of 2 with multiplications by even powers of 2 may be useful in other contexts.

A different approach is to decrease the number k of vectors x_i that are multiplied with $\mathbf{O}$. In our attacks, we combine leakage from all 9 vectors to increase the success probability of the attack. The fewer vectors are multiplied, the less information an attacker has available. In the tentative second-round parameters [8], the value of k in the MAYO_1 parameter set increases to 10, however, the

MAYO_2 parameter set (which has the same security level) remains at the smaller value of 4 (which reduces the size of the signature, at the cost of increasing the size of the public key). As mentioned previously, attackers are generally able to combine information from several signatures, thus even if we would set $k = 1$, an attacker that is capable of observing the signature generation for 9 signatures will have the same information as for a single signature with $k = 9$, so this may not be an effective countermeasure in practice.

11 Conclusion

We presented two practical single-trace power analysis-based side-channel attacks on an implementation of MAYO that are able to recover the full secret key of the scheme. Both attacks succeed with high probability, greater than 90%.

We also proposed countermeasures against the presented attacks and described an alternative implementation for one of the procedures targeted by the attacks that reduces the leakage by replacing multiplications with odd powers of 2 by multiplications with even powers of 2.

Our work demonstrates the importance of protecting modular multiplications against side-channel attacks. It also highlights opportunities for developing countermeasures by considering processor-specific behaviour and low-level optimisations. Future work includes developing stronger countermeasures against side-channel attacks on implementations of PQC algorithms, such as MAYO.

12 Acknowledgement

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation and by the Swedish Civil Contingencies Agency (Grant No. 2020-11632).

References

1. Arm: Arm v7-m architecture reference manual (2021), https://developer.arm.com/documentation/ddi0403/ee/
2. Aulbach, T., Campos, F., Krämer, J., Samardjiska, S., Stöttinger, M.: Separating oil and vinegar with a single trace: Side-channel assisted Kipnis-Shamir attack on UOV. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(3), 221–245 (June 2023). https://doi.org/10.46586/tches.v2023.i3.221-245
3. Aulbach, T., Kovats, T., Krämer, J., Marzougui, S.: Recovering Rainbow's secret key with a first-order fault attack. In: Batina, L., Daemen, J. (eds.) Progress in Cryptology - AFRICACRYPT 2022. pp. 348–368. Springer Nature Switzerland, Cham (2022). https://doi.org/10.1007/978-3-031-17433-9_15
4. Aulbach, T., Marzougui, S., Seifert, J.P., Ulitzsch, V.Q.: MAYo or MAY-not: Exploring implementation security of the post-quantum signature scheme MAYO against physical attacks. Workshop on Fault Diagnosis and Tolerance in Cryptography (September 2024), https://fdtc.deib.polimi.it/FDTC24/slides/FDTC2024-talk-2.2.pdf

5. Backlund, L., Ngo, K., Gärtner, J., Dubrova, E.: Secret key recovery attack on masked and shuffled implementations of CRYSTALS-Kyber and Saber. In: Zhou, J., Batina, L., Li, Z., Lin, J., Losiouk, E., Majumdar, S., Mashima, D., Meng, W., Picek, S., Rahman, M.A., Shao, J., Shimaoka, M., Soremekun, E., Su, C., Teh, J.S., Udovenko, A., Wang, C., Zhang, L., Zhauniarovich, Y. (eds.) Applied Cryptography and Network Security Workshops. pp. 159–177. Springer Nature Switzerland, Cham (2023)
6. Beullens, W.: Improved cryptanalysis of UOV and Rainbow. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 348–373. Springer International Publishing, Cham (2021)
7. Beullens, W.: MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In: AlTawy, R., Hülsing, A. (eds.) Selected Areas in Cryptography. pp. 355–376. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-99277-4_17
8. Beullens, W.: MAYO: Overview + Updates. NIST PQC Seminar (September 2024), https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/pqc-seminars/presentations/20-mayo-09242024.pdf
9. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: MAYO (June 2023), https://pqmayo.org/assets/specs/mayo.pdf
10. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: Nibbling MAYO: Optimized implementations for AVX2 and Cortex-M4. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(2), 252–275 (March 2024). https://doi.org/10.46586/tches.v2024.i2.252-275
11. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Advances in Cryptology - CRYPTO '99. vol. 1666, pp. 398–412. Springer (1999)
12. Chou, T., Kannwischer, M.J., Yang, B.: Rainbow on Cortex-M4. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(4), 650–675 (2021). https://doi.org/10.46586/TCHES.V2021.I4.650-675, https://doi.org/10.46586/tches.v2021.i4.650-675
13. Ding, J., Yang, B.Y., Chen, C.H.O., Chen, M.S., Cheng, C.M.: New differential-algebraic attacks and reparametrization of Rainbow. In: Bellovin, S.M., Gennaro, R., Keromytis, A., Yung, M. (eds.) Applied Cryptography and Network Security. pp. 242–257. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
14. Furue, H., Kiyomura, Y., Nagasawa, T., Takagi, T.: A new fault attack on UOV multivariate signature scheme. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryptography. pp. 124–143. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-17234-2_7
15. Gringiani, A., Meneghetti, A., Signorini, E., Susella, R.: MAYO: Optimized implementation with revised parameters for ARMv7-m. Cryptology ePrint Archive, Paper 2023/540 (2023), https://eprint.iacr.org/2023/540
16. Guo, Q., Johansson, T., Nilsson, A.: A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 359–386. Springer International Publishing, Cham (2020)
17. Hashimoto, Y., Takagi, T., Sakurai, K.: General fault attacks on multivariate public key cryptosystems. In: Yang, B.Y. (ed.) Post-Quantum Cryptography. pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_1
18. Jendral, S., Dubrova, E.: MAYO key recovery by fixing vinegar seeds. Cryptology ePrint Archive, Paper 2024/1550 (2024), https://eprint.iacr.org/2024/1550

19. Jendral, S., Ngo, K., Wang, R., Dubrova, E.: Breaking SCA-protected CRYSTALS-Kyber with a single trace. In: 2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 70–73 (2024). https://doi.org/10.1109/HOST55342.2024.10545390
20. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99. pp. 206–222. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
21. Kipnis, A., Shamir, A.: Cryptanalysis of the oil and vinegar signature scheme. In: Krawczyk, H. (ed.) Advances in Cryptology — CRYPTO '98. pp. 257–266. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
22. Krämer, J., Loiero, M.: Fault attacks on UOV and Rainbow. In: Polian, I., Stöttinger, M. (eds.) Constructive Side-Channel Analysis and Secure Design. pp. 193–214. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-16350-1_11
23. Mus, K., Islam, S., Sunar, B.: Quantumhammer: A practical hybrid attack on the LUOV signature scheme. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. p. 1071–1084. CCS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3417272
24. National Institute of Standards and Technology: NIST announces additional digital signature candidates for the PQC standardization process (June 2023), https://csrc.nist.gov/news/2023/additional-pqc-digital-signature-candidates
25. National Institute of Standards and Technology: NIST announces 14 candidates to advance to the second round of the additional digital signatures for the post-quantum cryptography standardization process (October 2024), https://csrc.nist.gov/news/2024/pqc-digital-signature-second-round-announcement
26. Ngo, K., Dubrova, E., Johansson, T.: Breaking masked and shuffled CCA secure Saber KEM by power analysis. In: Proc. of the 5th Workshop on Attacks and Solutions in Hardware Security. pp. 51–61 (2021)
27. NIST: PQC standardization process: Announcing four candidates to be standardized, plus fourth round candidates. NIST Computer Security Resource Center (July 2022), https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4
28. Park, A., Shim, K., Koo, N., Han, D.: Side-channel attacks on post-quantum signature schemes based on multivariate quadratic equations - Rainbow and UOV -. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 500–523 (2018). https://doi.org/10.13154/TCHES.V2018.I3.500-523, https://doi.org/10.13154/tches.v2018.i3.500-523
29. Patarin, J.: The oil and vinegar signature scheme. In: Presented at the Dagstuhl Workshop on Cryptography September 1997 (1997)
30. Pébereau, P.: One vector to rule them all: Key recovery from one vector in UOV schemes. In: Saarinen, M.J., Smith-Tone, D. (eds.) Post-Quantum Cryptography. pp. 92–108. Springer Nature Switzerland, Cham (2024)
31. Pessl, P.: Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In: Dunkelman, O., Sanadhya, S.K. (eds.) Progress in Cryptology – INDOCRYPT 2016. pp. 153–170. Springer International Publishing, Cham (2016)
32. Pokorný, D., Socha, P., Novotný, M.: Side-channel attack on Rainbow post-quantum signature. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 565–568 (2021). https://doi.org/10.23919/DATE51398.2021.9474157

33. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 513–533. Springer International Publishing, Cham (2017)
34. Ravi, P., Roy, D.B., Bhasin, S., Chattopadhyay, A., Mukhopadhyay, D.: Number "not used" once - practical fault attack on pqm4 implementations of NIST candidates. In: Polian, I., Stöttinger, M. (eds.) Constructive Side-Channel Analysis and Secure Design. pp. 232–250. Springer International Publishing, Cham (2019)
35. Sayari, O., Marzougui, S., Aulbach, T., Krämer, J., Seifert, J.P.: HaMAYO: A fault-tolerant reconfigurable hardware implementation of the MAYO signature scheme. In: Wacquez, R., Homma, N. (eds.) Constructive Side-Channel Analysis and Secure Design. pp. 240–259. Springer Nature Switzerland, Cham (2024)
36. Shim, K.A., Koo, N.: Algebraic fault analysis of UOV and Rainbow with the leakage of random vinegar values. IEEE Transactions on Information Forensics and Security **15**, 2429–2439 (2020). https://doi.org/10.1109/TIFS.2020.2969555
37. Yi, H., Li, W.: On the Importance of Checking Multivariate Public Key Cryptography for Side-Channel Attacks: The Case of enTTS Scheme. The Computer Journal **60**(8), 1197–1209 (02 2017). https://doi.org/10.1093/comjnl/bxx010, https://doi.org/10.1093/comjnl/bxx010
38. Yi, H., Nie, Z.: Side-channel security analysis of UOV signature for cloud-based Internet of Things. Future Generation Computer Systems **86**, 704–708 (2018). https://doi.org/https://doi.org/10.1016/j.future.2018.04.083, https://www.sciencedirect.com/science/article/pii/S0167739X18304151