Splitting the MAYO: A Component-Wise Fault Injection Attack on Randomized MAYO

Mohamed Abdelmonem[1,2], Lejla Batina[2,3], Durba Chatterjee[3], Vincent Dankbaar[3], and Håvard Raddum[1]

[1] Simula UiB, Bergen, Norway, mohameda,haavardr@simula.no
[2] University of Bergen, Bergen, Norway
[3] Radboud University, Nijmegen, The Netherlands,
lejla.batina,durba.chatterjee,vincent.dankbaar@ru.nl

Abstract. We present a practical component-wise fault injection attack against randomized MAYO implementations. The attack targets the final addition of oil and vinegar components during signing. Depending on compiler optimization, a fault may expose either oil coefficients directly or vinegar coefficients that can be used to recover the oil coefficient via the public verification algorithm. By accumulating partial coefficient-wise information across multiple faulty signatures and exploiting the linear structure of the oil space, we recover the full secret oil space. Unlike prior attacks, our method does not rely on deterministic signing, memory reuse, or faulting a hash function. We validate the attack on ARM Cortex-M4 platforms using clock glitching and electromagnetic fault injection, and recover the MAYO-1 secret oil space for both the `-O3` and `-Os` compiler optimization settings. Additionally, we analyze several countermeasures against our attack, showing that some are ineffective or increase the attack cost, while others provide effective protection at additional implementation cost.

Keywords: fault injection attack, MAYO, multivariate cryptography, post-quantum cryptography

1 Introduction

The National Institute of Standards and Technology (NIST) finalized the first round of its standardization process for Post-Quantum Cryptography (PQC), selecting cryptographic schemes suitable for secure applications against quantum computing threats. Among the selected standards are the digital signature schemes ML-DSA [22] and SLH-DSA [23]. To broaden the scope of viable PQC solutions, NIST initiated a second PQC competition in September 2022, focusing on identifying additional signature algorithms offering distinct security assumptions and performance trade-offs relative to ML-DSA and SLH-DSA.

The MAYO signature scheme [6] emerged as a prominent candidate in the new call for PQC signatures, and has advanced to Round 3 [1]. MAYO belongs to the class of multivariate cryptographic schemes, whose security relies on the

hardness of solving systems of nonlinear multivariate quadratic (MQ) equations over finite fields. MAYO has short key sizes as well as small signature sizes, making it potentially suitable for resource-constrained embedded devices.

Several works have presented optimized implementations on software [7] as well as hardware platforms [15, 24], addressing the challenges of efficient and secure implementation on embedded devices. While MAYO offers theoretical assurances of security owing to the hardness of the MQ problem, its practical deployment necessitates a thorough assessment of implementation-level vulnerabilities. As these schemes are increasingly targeted for embedded applications, physical proximity becomes a realistic threat.

In this context, it becomes critical to assess how secure these implementations are against physical attacks, such as side-channel analysis and fault injection, in which the attacker targets the implementation rather than the underlying mathematical problem. Side-channel analysis (SCA) attacks are passive in nature, wherein an adversary observes physical characteristics during the execution to infer sensitive information about the cryptographic algorithm. In contrast, fault injection (FI) attacks are active in nature, as an adversary can tamper with or influence the execution via probing the power supply, e.g. by perturbing the clock signal or injecting electromagnetic (EM) glitches on the chip. Several successful SCA and FI attacks on PQC implementations reported in the literature [10] highlight a persistent gap between theoretical resilience and practical robustness.

Significant research has been conducted on the physical security evaluation of implementations of multivariate cryptographic schemes. The study of fault attacks on multivariate signature schemes began with [14]. It was later expanded to target schemes like UOV and Rainbow in [20, 25, 3, 12]. MAYO implementations have also been analyzed in this context [24, 4, 17, 2, 5]. However, existing works fall short in practice: they either rely predominantly on theoretical analyses, assume restrictive conditions for successful exploitation such as memory reuse or deterministic signatures, or target auxiliary components (e.g., hash functions) rather than the core signing procedure [17]. As a result, the practical fault resilience of randomized MAYO implementations remains insufficiently understood.

In this paper, we close this gap by presenting a practical fault injection attack on the second-round variant of MAYO [6]. Our attack exploits a fundamental vulnerability in the MAYO construction to enable secret key recovery—even in the randomized variant, where fresh randomness is used during each signing to produce different signatures for the same message. We systematically gather information from multiple faulty signatures to effectively recover the oil space. We demonstrate the attack end-to-end on an ARM Cortex-M4 microcontroller (commonly used in embedded applications). Furthermore, we show that simple countermeasures, such as comparing signatures against vinegar vectors, are ineffective, and that shuffling only increases the attack cost, while sign-then-verify and masking prevent or substantially raise the cost of the attack at the price of additional implementation overhead.

This work is inspired by the idea of *signature correction*, which was first introduced in the context of attacks on Dilithium [16] and later refined in [19]. It has also been applied to CROSS [18]. We apply this concept to MAYO and validate the attack in practice on a microcontroller using clock glitching and EMFI. While signature correction attacks were introduced for lattice-based schemes, they do not directly extend to randomized multivariate schemes. In randomized MAYO, component-wise faults recover isolated coefficients originating from independently sampled oil vectors that cannot be directly concatenated into a valid secret vector. Our attack overcomes this algebraic barrier without requiring faulting of hash functions or assumptions on memory reuse.

Contributions. The main contributions of this work are:

1. We demonstrate a practical fault attack targeting coefficient-wise addition of oil and vinegar vectors, that exploits the fact that the final o coefficients of the vinegar vectors are always zero. Since the oil space has dimension o, linear algebra allows efficient reconstruction of the complete oil space, thereby revealing the secret key.
2. We execute the fault injection attack in practice using two different FI techniques against a randomized MAYO implementation from the pqm4 repository. Clock glitching was performed on a CW308-STM32F3 board with an ARM Cortex-M4 STM32F303RBT6 processor using a ChipWhisperer Husky. EMFI was performed on a Nucleo-L4R5ZI board with an STM32L4R5ZIT6PU processor using a Keysight EMFI Transient Probe.
3. We show that existing countermeasures proposed in prior works are either completely ineffective against our attack or can be circumvented with only minor modifications to the original attack strategy, or come with significantly increased cost.

Paper Outline. The remainder of this paper is structured as follows. In Section 2, we provide a brief introduction to the MAYO signature scheme and other relevant background regarding known attack strategies. Section 3 presents our proposed fault injection and signature correction attack in detail, including formal descriptions and cost estimation. Section 4 presents the experimental setup details and results of the fault injection attack in practice. Section 5 discusses potential countermeasures against our attack and evaluates their effectiveness. Finally, we summarize our findings in Section 6.

2 Preliminaries

2.1 Notation and Basic Definitions

Let $\mathbb{F}_q$ denote the finite field containing q elements. Vectors of length n over $\mathbb{F}_q$ are denoted as $\mathbb{F}_q^n$ and represented using bold lowercase letters (e.g., $\mathbf{x}, \mathbf{y}$), while matrices over $\mathbb{F}_q$ use bold uppercase letters (e.g., $\mathbf{A}, \mathbf{B}$). The i-th component of vector $\mathbf{x}$ is indicated as $\mathbf{x}[i]$, with indices starting at zero. For indices $0 \leq$

$i < j \leq n$, we write $\mathbf{x}[i:j] \in \mathbb{F}_q^{j-i}$ to represent the subvector comprising the elements $x_i, x_{i+1}, \ldots, x_{j-1}$. If $\mathbf{A}$ is a matrix then $\mathbf{A}[i:]$ indicates the i-th row and $\mathbf{A}[:j]$ the j-th column of $\mathbf{A}$. The concatenation of vectors $\mathbf{u} \in \mathbb{F}_q^a$ and $\mathbf{v} \in \mathbb{F}_q^b$ is denoted by $\mathbf{u}\|\mathbf{v} \in \mathbb{F}_q^{a+b}$.

Multivariate Quadratic Map: A **multivariate quadratic map** $\mathcal{P}$ over the finite field $\mathbb{F}_q$ consists of m quadratic polynomials in n variables. Each component $p_i(\mathbf{x})$, where $\mathbf{x} = (x[0], \ldots, x[n-1])$, is a multivariate polynomial of total degree at most two with coefficients in $\mathbb{F}_q$. Together, the components form the map $\mathcal{P}$, which defines a function

$$\mathcal{P}(\mathbf{x}) = (p_0, \ldots, p_{m-1}) : \mathbb{F}_q^n \to \mathbb{F}_q^m.$$

The **differential** of $\mathcal{P}$ is a bilinear map $\mathcal{P}' : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q^m$ and is given by:

$$\mathcal{P}'(\mathbf{x}, \mathbf{y}) = \mathcal{P}(\mathbf{x} + \mathbf{y}) - \mathcal{P}(\mathbf{x}) - \mathcal{P}(\mathbf{y}). \tag{1}$$

When the degree of $\mathcal{P}$ is 2, the differential $\mathcal{P}'$ is a linear mapping when only one of $\mathbf{x}$ or $\mathbf{y}$ is considered as a variable.

The **Unbalanced Oil and Vinegar (UOV) trapdoor function** is a multivariate quadratic map $\mathcal{P}$ that vanishes on a secret linear subspace $\mathcal{O} \subset \mathbb{F}_q^n$ of dimension $\dim(\mathcal{O}) = m$, i.e.,

$$\mathcal{P}(\mathbf{o}) = 0 \quad \text{for all } \mathbf{o} \in \mathcal{O}.$$

This linear subspace is called the **oil space** and serves as the trapdoor information. We define $\mathbf{O} \in \mathbb{F}_q^{n \times m}$ as a matrix whose columns form a basis of $\mathcal{O}$. We choose the basis matrix, so it is always given in systematic form, meaning it has the structure:

$$\mathbf{O} = \begin{pmatrix} \bar{\mathbf{O}} \\ \mathbf{I}_m \end{pmatrix},$$

where $\mathbf{I}_m$ is the $m \times m$ identity matrix and $\bar{\mathbf{O}} \in \mathbb{F}_q^{(n-m) \times m}$.

Knowing this trapdoor information, it is easy to reconstruct a preimage $\mathbf{x} \in \mathbb{F}_q^n$ of a given salted hash digest $\mathbf{t} \in \mathbb{F}_q^m$ such that $\mathcal{P}(\mathbf{x}) = \mathbf{t}$.

This is done by randomly choosing a vinegar vector $\mathbf{v} \in \mathbb{F}_q^n$ and solving for an oil vector $\mathbf{o} \in \mathcal{O}$ such that $\mathcal{P}(\mathbf{v}+\mathbf{o}) = \mathbf{t}$. Using the differential of $\mathcal{P}$ as defined in (1), we can write

$$\mathbf{t} = \mathcal{P}(\mathbf{v} + \mathbf{o}) = \mathcal{P}'(\mathbf{v}, \mathbf{o}) + \mathcal{P}(\mathbf{o}) + \mathcal{P}(\mathbf{v}). \tag{2}$$

Since $\mathcal{P}(\mathbf{o}) = 0$ by construction and $\mathcal{P}(\mathbf{v})$ is a fixed constant and $\mathcal{P}'(\mathbf{v}, \mathbf{o})$ is linear in $\mathbf{o}$, this yields a system of m linear equations in m variables, which can be solved efficiently. The signature is then given by $\mathbf{s} = \mathbf{v} + \mathbf{o}$. The verifier, using the public map $\mathcal{P}$ and the hash digest $\mathbf{t}$, checks whether $\mathcal{P}(\mathbf{s}) = \mathbf{t}$ holds to validate the signature.

2.2 The MAYO Signature Scheme

MAYO is a post-quantum digital signature algorithm based on the UOV framework, designed to offer efficient signatures resistant to both classical and quantum attacks [6]. The primary limitation of traditional UOV signatures is their large key sizes. MAYO addresses this issue by adopting an oil space $\mathcal{O} \subset \mathbb{F}_q^n$ with reduced dimension $o := \dim(\mathcal{O}) < m$, so that the basis matrix becomes

$$\mathbf{O} = \begin{pmatrix} \bar{\mathbf{O}} \\ \mathbf{I}_o \end{pmatrix},$$

with $\bar{\mathbf{O}} \in \mathbb{F}_q^{(n-o)\times o}$. This significantly decreases the key sizes, but it also implies that the resulting linear system of equations,

$$\mathcal{P}(\mathbf{v} + \mathbf{o}) = \mathbf{t},$$

often becomes unsolvable, as it comprises m equations in only o unknowns, where $o < m$. To mitigate this issue, MAYO employs a construction that expands the oil space by a factor of k, thus enabling solutions within a higher-dimensional space. This is achieved by defining a new map:

$$\mathcal{P}^*(\mathbf{y}_0, \ldots, \mathbf{y}_{k-1}) : \mathbb{F}_q^{kn} \to \mathbb{F}_q^m,$$

which "whips" the original UOV map $\mathcal{P}$ into a larger one that operates on k vinegar-oil pairs simultaneously. The map is defined as

$$\mathcal{P}^*(\mathbf{y}_0, \ldots, \mathbf{y}_{k-1}) := \sum_{i=0}^{k-1} \mathbf{E}_{ii}\mathcal{P}(\mathbf{y}_i) + \sum_{i=0}^{k-1}\sum_{j=i+1}^{k-1} \mathbf{E}_{ij}\mathcal{P}'(\mathbf{y}_i, \mathbf{y}_j),$$

where $\mathbf{E}_{ij} \in \mathbb{F}_q^{m\times m}$ are fixed public matrices and $\mathcal{P}'$ is the differential of $\mathcal{P}$. By choosing parameters such that $ko \geq m$, the new space $\mathcal{O}^k = \{(\mathbf{o}_0, \ldots, \mathbf{o}_{k-1}) \mid \mathbf{o}_i \in \mathcal{O}\}$ becomes large enough to ensure that the signer can find a preimage under $\mathcal{P}^*$ with high probability. The signature consists of k oil vectors masked with vinegar values $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{k-1} \in \mathbb{F}_q^{n-o}$:

$$\mathbf{s}_i = \mathbf{v}_i || \mathbf{0} + \mathbf{o}_i = (\mathbf{v}_i + \bar{\mathbf{O}}\mathbf{x}_i) || \mathbf{x}_i, \tag{3}$$

where $\mathbf{x}_i \in \mathbb{F}_q^o$, $\mathbf{o}_i \in \mathcal{O}$, and $i \in \{0, \ldots, k-1\}$. The parameter sets of the round 2 version of MAYO are given in Table 1.

MAYO has two versions: deterministic and randomized. In the deterministic version, the vinegar vectors are derived deterministically from the message to be signed, so signing the same message twice will result in the same vinegar vectors. In the randomized version, the vinegar vectors depend on randomness introduced during the signing process, so signing the same message multiple times yields different vinegar vectors and signatures each time. The complete MAYO signing algorithm is given in Algorithm 1.

Table 1: Parameter sets of MAYO.

Parameter set	**MAYO-1**	**MAYO-2**	**MAYO-3**	**MAYO-5**
Security level	1	1	3	5
n	86	81	118	154
m	78	64	108	142
o	8	17	10	12
k	10	4	11	12
q	16	16	16	16

Algorithm 1 MAYO.Sign(sk, M) [6]

Input: Secret key sk, message M
Output: Signature σ

$(\text{seed}_{sk}, \mathbf{O}) \leftarrow \text{sk}$
$\text{seed}_{pk} \leftarrow \text{SHAKE256}(\text{seed}_{sk})$
$P^*(\mathbf{x}_0, \ldots, \mathbf{x}_{k-1}) \leftarrow \sum_{i=0}^{k-1} E_{ii}\mathcal{P}(\mathbf{x}_i) + \left(\sum_{0 \le i < j \le k-1} E_{ij}\mathcal{P}'(\mathbf{x}_i, \mathbf{x}_j)\right)$
$R \leftarrow \{0,1\}^r$ ▷ *Deterministic variant:* $R \leftarrow \{0\}^r$
$\text{salt} \leftarrow \text{SHAKE256}(M \| R \| \text{seed}_{sk})$
$t \leftarrow \text{SHAKE256}(M \| \text{salt})$
for *ctr* from 0 to 255 **do**
    $\mathbf{V} \leftarrow \text{SHAKE256}(M \| \text{salt} \| \text{seed}_{sk} \| ctr)$
    $\mathbf{v}_0, \ldots, \mathbf{v}_{k-1} \leftarrow \text{Decode}(\mathbf{V})$
    $(\mathbf{A}, \mathbf{y}) \leftarrow \text{BuildLinearSystem}(\{\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}\}, \mathbf{O}, P^*, \mathbf{t})$
    $\mathbf{x} \leftarrow \text{SampleSolution}(\mathbf{A}, \mathbf{y})$ ▷ *Try to find* $A\mathbf{x} = \mathbf{y}$ *(i.e.* $\mathcal{P}^*(s) = t$*)*
    if $\mathbf{x} \neq \perp$ **then break**
    end if
end for
$s \leftarrow \{(\mathbf{v}_i + \bar{\mathbf{O}}\mathbf{x}_i) \| \, x_i\}_{0 \le i \le k-1}$
return $\sigma = (s, \text{salt})$

2.3 Key Recovery from a Single Oil Vector

If an attacker knows one oil vector, recovering the entire oil space is merely a matter of solving linear equations by using the reconciliation attack, which was developed against the Rainbow Signature [9] and it also works on MAYO [6].

2.4 Signature Correction Attacks

Signature correction attacks have been introduced to attack the Dilithium scheme [16, 19]. The core idea behind these attacks is to induce a fault during the signing process, resulting in a faulty signature that omits a portion of the secret. The attacker then attempts to correct the faulty signature by exhaustively guessing the missing part and checking each candidate using the public verification algorithm. Since the verification procedure reveals whether the correction was successful, it effectively serves as an oracle, allowing the attacker to recover the missing secret information.

2.5 Existing Fault Attacks against MAYO

Fault injection (FI) attacks on the MAYO signature scheme were previously discussed in [17, 4, 24, 5, 2]. In [24], the authors propose two scenarios for faulting the signing algorithm, both of which lead to key recovery. In the first scenario, they assume that vinegar vectors are initialized to a constant known value before sampling, which they set to zero for simplicity. A loop-abort fault in this case results in one or more vinegar vectors retaining this initial constant. If the sampling of the $(k-1)$-th vinegar vector is skipped, the resulting faulted signature is

$$\mathbf{S}' = (\mathbf{s}_0, \ldots, \mathbf{s}'_{k-1}) = ((\mathbf{v}_0||\mathbf{0}) + \mathbf{o}_0, \ldots, \mathbf{0} + \mathbf{o}_{k-1}).$$

Since $\mathbf{s}'_{k-1} = \mathbf{o}_{k-1}$, we have therefore found one oil vector and the whole oil space can be recovered using the reconciliation attack [9].

The second scenario is a differential fault attack that only works in the deterministic setting. The attacker first obtains a correctly computed signature $\mathbf{S} = (\mathbf{s}_0, \ldots, \mathbf{s}_{k-1})$, with

$$\mathbf{s}_{k-1} = (\mathbf{v}_{k-1}||\mathbf{0}) + \mathbf{o}_{k-1},$$

then injects a loop-abort fault during the computation of a second signature $\mathbf{S}'$ where

$$\mathbf{s}'_{k-1} = (\mathbf{v}_{k-1}||\mathbf{0}) + \mathbf{0}.$$

Taking the difference reveals one of the oil vectors

$$\mathbf{s}_{k-1} - \mathbf{s}'_{k-1} = \mathbf{o}_{k-1} \in \mathcal{O}.$$

This attack only works in the deterministic version, where the vinegar only depends on the message. In the randomized version, we have different vinegar vectors even for the same message, so this attack will not work.

In [4], the authors argue that this attack may also succeed in the randomized mode, provided at least one vinegar value in the faulted signature matches a vinegar value in the correct signature. This scenario can occur if the memory holding the vinegar vectors is allocated on the stack without being properly initialized prior to sampling. Consequently, vinegar vectors might retain values from the previous signature generation process.

In [2], the authors propose disturbing the setup of the linear system in line 10 of Algorithm 1, which leads to an alternative solution $\mathbf{x}'_{k-1}$. Taking the difference between a correct signature and a faulty signature yields

$$\begin{aligned}\mathbf{s}'_{k-1} - \mathbf{s}_{k-1} =& (\mathbf{v}_{k-1} + \bar{\mathbf{O}}\mathbf{x}'_{k-1})||\mathbf{x}'_{k-1} - (\mathbf{v}_{k-1} + \bar{\mathbf{O}}\mathbf{x}_{k-1})||\mathbf{x}_{k-1} \\ =& \bar{\mathbf{O}}(\mathbf{x}'_{k-1} - \mathbf{x}_{k-1})||(\mathbf{x}'_{k-1} - \mathbf{x}_{k-1}) \in \mathcal{O}.\end{aligned}$$

Since this approach relies on the vinegar vector $\mathbf{v}_{k-1}$ being identical in both signatures, the attack requires a deterministic signing setting.

In [5], the authors present an alternative method for obtaining a different solution $\mathbf{x}'_{k-1}$, which can be exploited in the same manner. They fault the target

Table 2: Comparison of fault attacks on the MAYO signature scheme.

Source	Attack description	Assumption	# faulty sigs	Rand. mode?	Evaluation
[24]	Skip vinegar sampling	known vinegar initialization value	1	✓	Theoretical
	Skip oil-to-vinegar addition	None	1	✗	Theoretical
[4]	Fix vinegar vector	Zero-initialization/ exact memory reuse	1	✓	Practical
[2]	Disturb linear-system setup	None	1	✗	Theoretical
[5]	Fault target vector/ solution sampling	Unprotected hashing	1	✗	Practical
[17]	Fix vinegar seeds	Unprotected hashing	1	✓	Practical
This work	Skip vinegar-to-oil/ oil-to-vinegar addition, componentwise	None	Multiple	✓	Practical

vector t in line 6 of Algorithm 1 by attacking the SHAKE algorithm, again resulting in a distinct solution $\mathbf{x}'_{k-1}$. The attack is demonstrated using a clock-glitching fault injection, but it is likewise limited to the deterministic setting.

A similar attack on the SHAKE algorithm was introduced in [17]. Here, the authors attack the SHAKE algorithm, which computes the seed used to sample the vinegar values in line 8 of Algorithm 1 instead. As a result, the faulty vinegar vector becomes a constant known to the attacker. Both approaches assume that the hash function used to compute the target vector t or the vinegar seeds can be faulted. However, countermeasures against such attacks have been proposed, for instance by protecting the hashing process with a fault-tolerant Keccak implementation, as described in [13, 21].

In this work, we present a fault injection attack on randomized MAYO that does not require any memory assumptions and does not require the ability to fault the hashing algorithm used. Our attack assumes the vinegar vectors are unknown quantities, freshly and independently generated for each call to the signature algorithm. We demonstrate that this attack can be carried out using both clock glitching and EMFI, and show that existing countermeasures are either insufficient to prevent this fault attack, can be easily circumvented with slight adjustments, or are expensive to implement. In Table 2, we compare existing attacks with our newly proposed attack.

3 Proposed Fault Injection Attack against MAYO

The idea behind our attack is to expose either a single or a small group of vinegar or oil coefficients. In practice, the exact attack strategy depends on the compiler optimization level, since different levels yield different instruction sequences and, consequently, different fault targets. In this section, we present the case where we assume that a single coefficient of a vinegar vector can be revealed. Moreover, we also first assume that every fault injection is successful. How to handle the more realistic scenario where some faults are unsuccessful will be explained in Section 3.3.

Our proposed attack consists of three main steps:

1. **Fault Injection:**
   The goal of this step is to extract partial information about secret oil vectors by faulting the addition of oil and vinegar coefficients. Depending on the implementation and the induced fault, either the vinegar coefficient or the oil coefficient itself is exposed.
2. **Signature Correction:**
   If the vinegar coefficient is exposed, we use the verification algorithm as an oracle to recover the corresponding oil coefficient. If the oil coefficient is exposed directly, this step is not required.
3. **Recovery of Oil Space:**
   In both cases, the result is a known oil coefficient together with the corresponding $\mathbf{x} \in \mathbb{F}_q^o$. We use these pairs to reconstruct the basis of the oil space as described in Section 3.2.

We describe the three steps in detail in the following subsections. The explanation below first considers the vinegar-revealing case, for which signature correction is required. We then explain how the same oil-space reconstruction applies when the oil coefficient is revealed directly. The steps can be done either in a fully serialized way or in an interleaved way, where single rows of the basis matrix $\bar{\mathbf{O}}$ are recovered as soon as we have enough information to do so. The explanation below follows the second approach, recovering one row of $\bar{\mathbf{O}}$ for every o faulted signatures.

3.1 Fault Injection and Signature Correction

Our fault attacks target the addition of one oil and vinegar vector in line 15 of Algorithm 1. It suffices to target just one of the signature vectors in

$$\mathbf{S} = (\mathbf{s}_0, \ldots, \mathbf{s}_{k-1}),$$

for example

$$\mathbf{s}_0 = (\mathbf{v}_0 + \bar{\mathbf{O}}\mathbf{x}_0)||\mathbf{x}_0.$$

As we only need to focus on a single signature vector and the specific index is irrelevant, we will omit the subscript from this point onward and simply write $\mathbf{s}, \mathbf{v}, \mathbf{o}$ instead of $\mathbf{s}_0, \mathbf{v}_0, \mathbf{o}_0$:

$$\mathbf{s} = (\mathbf{v} + \bar{\mathbf{O}}\mathbf{x})||\mathbf{x}. \tag{4}$$

What has been done so far in [24, 4] is to induce a fault that skips the addition of the oil vector entirely. This results in a faulty signature of the form $\mathbf{s}' = \mathbf{v} \,\|\, \mathbf{0}$, where the oil part is set to zero. Instead of signing again and applying a differential attack, one can attempt to recover the correct signature by correcting the faulty one. This is done by testing all possible oil candidates $\boldsymbol{\alpha} \in \mathbb{F}_q^n$ and computing

$$\mathbf{s}'' = \mathbf{v} \,\|\, \mathbf{0} + \boldsymbol{\alpha}.$$

For each candidate, one can use the verification algorithm to check whether the modified signature verifies correctly. If verification succeeds for some $\boldsymbol{\alpha}$, we conclude that $\boldsymbol{\alpha}$ is a valid oil vector. Since a single such vector is sufficient to recover the entire oil space, as shown in [9], this would break the scheme. However, this naive approach requires testing up to q^{n-o} possible $\boldsymbol{\alpha}$'s, making the attack computationally infeasible for the parameter sizes used in MAYO.

A natural refinement is to inject a fault that affects only a single coefficient of the signature vector $\mathbf{s}$, and attempt to correct it by guessing a single value $\alpha_i \in \mathbb{F}_q$ instead of an entire vector. Specifically, we inject the fault and correct it in only one component, to get

$$\mathbf{s}''[i] = \mathbf{v}[i] + \alpha_i$$

for some index $i \in \{0, \ldots, n-o-1\}$. This reduces the complexity of the attack to at most q verification queries. However, it does not recover the full oil vector; instead, it only reveals the value of a single coefficient of some vector in the oil space:

$$\mathbf{o}[i] = \alpha_i,$$

whereas the remaining components of the oil vector remain unknown.

At first glance, it may seem that repeating this process for all indices i would allow recovery of the entire oil vector. However, each signature uses a freshly sampled oil vector drawn from the same oil subspace. As a result, each recovered coordinate α_i originates from a different oil vector. Concatenating these values does not necessarily yield a vector within the oil space, and therefore does not lead to recovery of the secret. Note, however, that the $\mathbf{x}$ used to produce each oil vector is given directly in the signature and can be exploited. We follow the three steps outlined below and explain in the next section how this approach can be used to recover the entire oil space.

1. **Inject faults to create defective signatures.**
   For each $i \in \{0, \ldots, n-o-1\}$, perform a fault injection which skips the addition of the oil vector's coefficient i. This produces faulty signatures $\mathbf{s}'$ where
   $$\mathbf{s}'[j] = \begin{cases} \mathbf{v}[j] \text{ for } j = i \\ \mathbf{v}[j] + \mathbf{o}[j] \text{ for } j \neq i. \end{cases}$$
2. **Recover oil components using the verifier as an oracle.**
   For each faulty signature $\mathbf{s}'$, test values $\alpha \in \mathbb{F}_q$ such that $\mathbf{s}''$ where
   $$\mathbf{s}''[j] = \begin{cases} \mathbf{s}'[j] + \alpha \text{ for } j = i \\ \mathbf{s}'[j] \text{ for } j \neq i \end{cases}$$
   verifies successfully. When this happens, we conclude
   $$\alpha = \mathbf{o}[i]$$
   for the particular oil vector $\mathbf{o}$ chosen during signing.

3. **Accumulate partial oil vectors.**
   Repeat this process o times for each index i to end up with $o(n-o)$ partial oil vectors, each revealing the i-th component. For each of these partial oil vectors, we also save the last o components of the signature, namely the $\mathbf{x} \in \mathbb{F}_q^o$ used to produce the particular oil vector chosen in each signature.

Ineffective faults. A special case occurs when the targeted oil coefficient is zero. If the addition of this coefficient is skipped, then

$$s'[i] = v[i] = v[i] + o[i] = s[i],$$

and the resulting signature is identical to a correctly generated signature at the targeted position. We refer to such an event as an *ineffective fault*. An injection that has no effect on the computation produces the same observable outcome: the resulting signature already verifies without any correction.

Consequently, if a faulty signature verifies without modification, the attacker cannot distinguish whether an ineffective fault occurred or whether the injection had no effect. Interpreting such an observation as the correction $\alpha = 0$ could therefore introduce an incorrect equation into the oil-space recovery. We hence discard signatures that already verify without modification. For the remaining vinegar-revealing faulty signatures, the correction search is restricted to $\alpha \in \mathbb{F}_q \setminus \{0\}$.

Direct exposure of oil coefficients. The correction step above is only required when the fault leaves the vinegar coefficient $\mathbf{v}[i]$ in the faulty signature. Depending on the instruction sequence and the induced fault, it is also possible that the oil coefficient itself is exposed. In this case, the faulty signature directly reveals

$$\alpha_i = \mathbf{o}[i] = \bar{\mathbf{O}}[i:]\mathbf{x},$$

where $\mathbf{x}$ is again given by the final o components of the signature vector. No verification queries are required to correct this coefficient.

Thus, both fault outcomes provide exactly the same information for the oil-space recovery:

$$\big(\mathbf{x}, \alpha_i\big), \qquad \alpha_i = \bar{\mathbf{O}}[i:]\mathbf{x}.$$

The only difference is how α_i is obtained: for a vinegar-revealing fault it is recovered using the verification oracle, whereas for an oil-revealing fault it is obtained directly from the faulty signature. The reconstruction described in Section 3.2 is identical in both cases.

3.2 Recovery of Oil Space

When generating a faulty signature targeting index i and using the correction oracle, we obtain an oil vector whose coefficients at indices in $\{i, n-o, n-o+$

$1, \ldots, n-1\}$ are known. After finding o such oil vectors, we put them as column vectors in an $n \times o$ matrix

$$\mathbf{A}_i = \begin{bmatrix} * & \cdots & * \\ \vdots & & \vdots \\ a_{i,0} & \cdots & a_{i,o-1} \\ * & \cdots & * \\ \vdots & & \vdots \\ a_{n-o,0} & \cdots & a_{n-o,o-1} \\ \vdots & & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,o-1} \end{bmatrix}, \tag{5}$$

where $*$ indicates unknown values.

Denote by $\mathbf{X}_i$ the $o \times o$ submatrix of $\mathbf{A}_i$ consisting of the last o rows. When building the matrix $\mathbf{A}_i$, we ensure that $\mathbf{X}_i$ has full rank. This is done by checking if the $\mathbf{X}_i$-part of a new column to be added to $\mathbf{A}_i$ is linearly dependent on existing columns in $\mathbf{X}_i$. If it is, we reject the column and sample another one. After the complete $\mathbf{A}_i$ is constructed, we know that $\mathbf{X}_i$ is an invertible matrix.

Using $\mathbf{X}_i^{-1}$ to change the basis of the columns in $\mathbf{A}_i$, we get

$$\mathbf{A}_i\mathbf{X}_i^{-1} = \begin{bmatrix} * \\ \mathbf{A}_i[i:]\mathbf{X}_i^{-1} \\ * \\ \mathbf{I}_o \end{bmatrix}.$$

The columns in $\mathbf{A}_i\mathbf{X}_i^{-1}$ are therefore the same as in $\mathbf{O}$, so multiplying the entries in row i of $\mathbf{A}_i$ with $\mathbf{X}_i^{-1}$ recovers row i of the secret oil space: $\bar{\mathbf{O}}[i:] = \mathbf{A}_i[i:]\mathbf{X}_i^{-1}$.

Repeating this process for $0 \leq i < n - o$ will recover the basis for the entire secret oil space. The complete algorithm for recovering the oil space from corrected faulty signatures is given in Algorithm 2.

We show the attack procedure on a small example, where we choose a prime field for arithmetic convenience.

Example 3.1 $n = 6, o = 2, q = 17$
Let the secret oil space $\mathcal{O} \subset \mathbb{F}_q^n$ be defined by the basis matrix

$$\mathbf{O} = \begin{pmatrix} 10 & 7 \\ 11 & 3 \\ 2 & 2 \\ 3 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Using the verification oracle, assume we have recovered the following two partial vectors in the oil space.

Algorithm 2 Recovering oil space

Require: the ability to generate faulty MAYO signatures where the addition of any oil vector coefficient is skipped
Ensure: Basis of oil space $\mathcal{O}$

$M \leftarrow$ random message
$\bar{\mathbf{O}} \leftarrow []^{n-o \times o}$ ▷ Empty matrix of size $(n-o) \times o$
for $0 \leq i < n - o$ **do**
    $\mathbf{a} \leftarrow []^{o}$ ▷ Empty vector of length o
    $\mathbf{X} \leftarrow []^{o \times o}$ ▷ Empty matrix of size $o \times o$
    $t \leftarrow 0$
    while $t < o$ **do**
        $\mathbf{s} \leftarrow$ signature on M where addition of $\mathbf{o}[i]$ is skipped
        if verify$(M, \mathbf{s})$ returns true **then**
            continue
        end if
        $\mathbf{x} \leftarrow \mathbf{s}[n-o : n]$
        if $\mathbf{x}$ is linearly independent of existing columns in $\mathbf{X}$ **then**
            $\mathbf{s}' \leftarrow \mathbf{s}$
            for $\alpha \in \mathbb{F}_q \setminus \{0\}$ **do**
                $\mathbf{s}'[i] \leftarrow \mathbf{s}[i] + \alpha$
                if verify$(M, \mathbf{s}')$ returns true **then**
                    break
                end if
            end for
            $\mathbf{X}[: t] \leftarrow \mathbf{x}$
            $\mathbf{a}[t] \leftarrow \alpha$
            $t \leftarrow t + 1$
        end if
    end while
    $\mathbf{X}^{-1} \leftarrow$ inverse of $\mathbf{X}$
    $\bar{\mathbf{O}}[i :] \leftarrow \mathbf{a}\mathbf{X}^{-1}$
end for
return $\bar{\mathbf{O}}$

$$\mathbf{o}_0 = \begin{pmatrix} 16 \\ * \\ * \\ * \\ 6 \\ 1 \end{pmatrix}, \quad \mathbf{o}_1 = \begin{pmatrix} 2 \\ * \\ * \\ * \\ 13 \\ 6 \end{pmatrix}.$$

This gives

$$\mathbf{X}_0 = \begin{pmatrix} 6 & 13 \\ 1 & 6 \end{pmatrix}, \quad \mathbf{A}_0[0 :] = \begin{pmatrix} 16 & 2 \end{pmatrix}.$$

Inverting $\mathbf{X}_0$ *yields*

$$\mathbf{X}_0^{-1} = \begin{pmatrix} 1 & 12 \\ 14 & 1 \end{pmatrix}.$$

The multiplication $\mathbf{A}_0[0:]\mathbf{X}_0^{-1}$ *then gives*

$$\begin{pmatrix} 16 & 2 \end{pmatrix} \begin{pmatrix} 1 & 12 \\ 14 & 1 \end{pmatrix} = \begin{pmatrix} 10 & 7 \end{pmatrix},$$

which is equal to $\bar{\mathbf{O}}[0:]$.

3.3 Handling unsuccessful faults

In practice, an injected fault does not always produce the intended effect. In our attack, a fault is considered successful only if it exposes the targeted oil or vinegar coefficient while leaving the rest of the signature computation unaffected. Even when only the targeted index is affected, the resulting value may be random rather than the desired oil or vinegar value. In this section, we describe how to validate that a recovered oil row is indeed correct, even when successful faults occur only with probability $p < 1$.

As described in Section 3.1, signatures that already verify without modification are discarded. In the following, N denotes the number of signatures retained after this filtering step, and p denotes the probability that a retained observation corresponds to a successful fault.

For each target index, we generate a pool of N faulty signatures, where N depends on p in a way that will be explained later.

To recover the correct row despite unsuccessful faults, we do not rely on a single recovery attempt per row of the matrix defining the oil space. Instead, for every fixed index i, we repeatedly sample sets of o faulty signatures whose corresponding vectors are from the pool of N signatures. For each such set for which $\mathbf{X}_i$ is invertible, we compute a candidate row

$$\mathbf{r} = \mathbf{A}_i[i:]\mathbf{X}_i^{-1}.$$

Every combination of successfully faulted signatures will recover the same correct oil row $\bar{\mathbf{O}}[i:]$. In contrast, a combination containing at least one incorrectly faulted signature will, in general, produce an incorrect candidate row.

To distinguish the correct row from incorrect candidates, we evaluate each candidate $\mathbf{r}$ against all signatures in the pool. Let $(\mathbf{x}_j, \alpha_j)$ denote the information obtained from the j-th faulty signature, where $\mathbf{x}_j \in \mathbb{F}_q^o$ consists of the final o coefficients of the corresponding signature vector and $\alpha_j \in \mathbb{F}_q$ is the recovered value of the targeted oil coefficient, ideally satisfying $\alpha_j = \bar{\mathbf{O}}[i:]\mathbf{x}_j$. For a candidate row $\mathbf{r} \in \mathbb{F}_q^o$, we define its agreement with the pool as

$$\mathrm{agr}(\mathbf{r}) = \left|\{j \in \{1, \ldots, N\} : \mathbf{r}\mathbf{x}_j = \alpha_j\}\right|.$$

Let S denote the number of successful faults in the pool. If each retained observation is successful independently with probability p, then

$$S \sim \mathrm{Bin}(N, p), \quad \mathrm{agr}(\bar{\mathbf{O}}[i:]) \geq S,$$

since every successfully faulted signature yields a correct equation. The agreement value for the correct row is therefore at least as large as an S taken from the binomial distribution $\mathrm{Bin}(N, p)$.

To model the agreement of incorrect rows, we assume that the vectors $\mathbf{x}_j$ are independent and uniform over $\mathbb{F}_q^o$, and that the errors in the recovered coefficients are independent of these vectors and of one another. Under this random-error model, a fixed incorrect row satisfies an independently sampled equation with probability $1/q$. A candidate reconstructed from an invertible subset, however, already satisfies the o equations from which it was constructed. Conditional on this candidate being incorrect, each of the remaining $N - o$ observations agrees with it independently with probability $1/q$. Its agreement is therefore modeled by

$$W \sim o + \mathrm{Bin}\left(N - o, \frac{1}{q}\right).$$

In an actual attack, we do not construct rows for all possible $\binom{N}{o}$ subsets from the pool. Only K subsets are used, where K is chosen large enough to sample a set consisting of only successfully faulted signatures with high probability. A set with only successfully faulted signatures is called a *good* set and a set with one or more unsuccessfully faulted signatures is called a *bad* set. Assuming the realized pool contains S successful signatures, there are $M = \binom{N}{o}$ possible subsets, of which $G = \binom{S}{o}$ are good and $M - G$ are bad. If K distinct subsets are sampled uniformly (without replacement), the probability of sampling at least one good subset is

$$\Pr(\text{one good set among } K \text{ different sets}) = 1 - \frac{\binom{M-G}{K}}{\binom{M}{K}}. \tag{6}$$

In addition to sampling a good set, it is necessary to distinguish the correct row from candidates produced by bad sets via the agreement score. For that, we define an integer threshold t and accept a candidate $\mathbf{r}$ if $\mathrm{agr}(\mathbf{r}) \geq t$. The threshold must be low enough that the correct row reaches it with high probability, but high enough that incorrect candidates are very unlikely to reach it.

Since the agreement of the correct row is at least the number of successful faults, we have

$$\Pr[\mathrm{agr}(\bar{O}[i:]) \geq t] \geq \Pr[\mathrm{Bin}(N, p) \geq t]. \tag{7}$$

Thus, t should be chosen so that the right-hand side is high. Figure 1 illustrates this requirement together with the competing requirement that an incorrect candidate shall be very unlikely to reach the threshold.

The agreement values of candidates produced from bad sets are not necessarily independent, due to partially overlapping subsets chosen from the pool

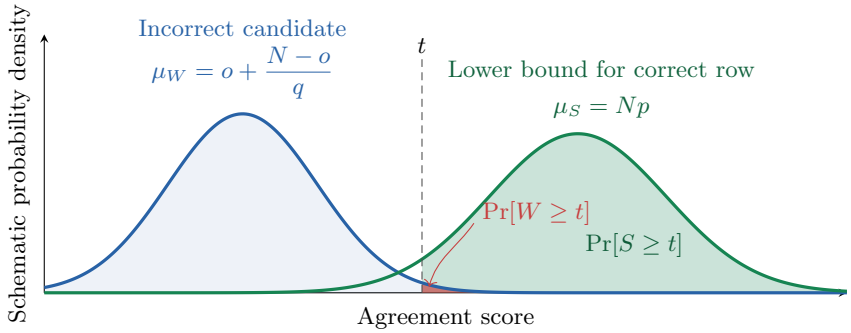


Fig. 1: Schematic illustration of the agreement-based distinguisher. The green curve represents the number of successful observations $S \sim \mathrm{Bin}(N, p)$, which provides a lower bound on the agreement score of the correct row, while the agreement score of a generated incorrect candidate is modeled by $W \sim o + \mathrm{Bin}(N - o, 1/q)$. The threshold t is chosen such that $S \geq t$ with high probability, while an incorrect candidate reaches the threshold only with very small probability. The green and red regions illustrate $\Pr[S \geq t]$ and $\Pr[W \geq t]$, respectively.

of N signatures. Nevertheless, the union bound does not require independence between candidates. Under the random-error model, if at most K subsets are sampled uniformly without replacement and independently of their observed values, we obtain

$$\begin{aligned} &\Pr(\text{at least one wrong row is accepted in K trials}) \\ \leq &K \cdot \Pr\left(o + \mathrm{Bin}\left(N - o, \frac{1}{q}\right) \geq t\right). \end{aligned} \tag{8}$$

Consequently, we choose the parameters N, K, and t such that the probability of accepting an incorrect row is very small. In particular, we require the false-acceptance bound in (8) to be below 10^{-4} per row for one pool of signatures and at most K trials. For **MAYO-1**, whose oil-space basis has 78 rows, this ensures the probability that no incorrect row is accepted during the recovery of the oil-space basis is at least 99%. This bound concerns the correctness of accepted rows and does not by itself guarantee that every row is recovered within the trial budget.

At the same time, we choose N and K such that the probability of sampling at least one good subset in K trials, as given in (6), exceeds 0.9 when estimating the number of correct signatures in a pool by $S \approx Np$. We deliberately allow a higher probability of recovery failure than of false acceptance. If no candidate reaches the agreement threshold within K trials, we discard the pool, collect a fresh one, and repeat the recovery procedure. Such a failure does not compromise the attack, whereas accepting an incorrect row would invalidate the reconstructed oil-space basis.

Thus, our parameter selection favors reliable identification of accepted rows over maximizing the probability of recovery from a single pool. The trial budget introduces a trade-off: increasing K improves the chance of sampling a usable good subset, but also increases the union-bound estimate of false acceptance. We therefore choose N, K, and t jointly based on the observed success rates and the desired reliability. The concrete parameter choices for clock glitching and EMFI are given in Sections 4.2 and 4.3, respectively, based on the value of p we observed in practice.

3.4 Cost analysis

In this section, we analyze the theoretical costs of our attack, assuming that skipping the addition is always successful, meaning we need only one attempt per index. The number of attempts needed in practice depends on the success rates and will be discussed in Section 4. The attack is divided into two main parts: the accumulation of partial information about the oil space and the recovery of the secret key. After successfully collecting the necessary oil vector information, the secret key can be efficiently recovered. This part consists of:

- inverting $n - o$ matrices, each of size $o \times o$.
- Executing $n - o$ vector/matrix multiplications, each involving vectors of length o and matrices of size $o \times o$.

For the largest MAYO variant, this translates to performing $2^{17.9}$ field multiplications to invert the matrices. These are small complexities and can be computed for all MAYO parameter sets within seconds.

To accumulate the necessary partial information from the oil space, the attack theoretically requires:

- $(n-o)o$ signatures generated with targeted fault injections. In practice, this number is slightly higher, as we may need to sample more than o faulty signatures before the matrices $\mathbf{X}_i$ have full rank. For $q = 16$ and $o = 8$, the expected number of vectors that need to be sampled until $\mathbf{X}_i$ has full rank is
$$\sum_{j=1}^{o} \frac{1}{1 - q^{-j}} \approx 8.07 \approx 1.009o.$$
This follows from the incremental rejection sampling in Algorithm 2, where a newly sampled vector is discarded whenever it is linearly dependent on the columns already contained in $\mathbf{X}_i$; (see [11, p. 35]). We have also verified experimentally that the $\mathbf{X}_i$ matrices we get in practice are almost always invertible. We therefore ignore this slight increase in the number of faulty signatures needed from here on.
- $q(n - o)o$ verification attempts in the worst-case scenario, to successfully correct the oil vector coefficients from the faulty signatures.

For the largest MAYO variant, this translates to running the verification oracle at most $2^{14.7}$ times, which has low practical complexity.

Table 3: Number of signatures and approximate number of verifications for MAYO and UOV for $\ell \in \{1, 4\}$.

		Signatures		Verifications				Signatures		Verifications	
MAYO	q	$\ell = 1$	$\ell = 4$	$\ell = 1$	$\ell = 4$	**UOV**	q	$\ell = 1$	$\ell = 4$	$\ell = 1$	$\ell = 4$
1	16	624	156	2^{12}	2^{22}	Ip	256	2992	748	2^{19}	2^{41}
2	16	1088	272	2^{13}	2^{23}	Is	16	6144	1536	2^{16}	2^{26}
3	16	1080	270	2^{13}	2^{23}	III	256	8064	2016	2^{20}	2^{42}
5	16	1704	426	2^{14}	2^{24}	V	256	14208	3552	2^{21}	2^{43}

3.5 Faulting multiple coefficients

Injecting faults into exactly one coefficient can be challenging in practical settings, depending on the fault injection method, and can even be impossible if the assembly code of an implementation adds 4 or 8 oil coefficients in the same instruction. For example, compiling the reference code with the `-O3` optimization, four coefficients are added together in the same instruction.

Note that our attack can still handle such cases, albeit at the expense of having to simultaneously guess multiple oil coefficients. If the addition of l oil coefficients is skipped, the corrected signature can be recovered by exhaustively testing all possible corrections in $\mathbb{F}_q^l$ until verification succeeds. While this strategy reduces the total number of required faulty signatures by a factor l, it exponentially increases the number of verification attempts per signature to q^l verifications for each faulty signature.

If $l = 4$, the complexity for accumulating the oil coefficients increases to

$$\frac{(n-o)o}{l} q^l \approx 2^{23.3}$$

for **MAYO-1**. This complexity is still practical. If the l affected coefficients are oil coefficients that are exposed directly, as in our `-O3` EMFI experiments in Section 4.3, no correction is necessary; one faulty signature then provides information for l rows of $\bar{\mathbf{O}}$ simultaneously. The q^l correction cost applies only when the l missing oil coefficients have to be recovered from exposed vinegar coefficients.

3.6 Transferability to UOV

The same attack can be applied to the original UOV algorithm submitted to the NIST competition (see line 11 of the signing algorithm in [8]). However, the correction step becomes more expensive because the field size q is larger for most parameter sets. In Table 3, we give an overview of how many signatures and expected verifications are needed for MAYO and UOV to perform this attack over all security parameters, assuming we skip the addition of $l = 1$ or $l = 4$ indices at a time.

4 Experimental Results

In this section, we first describe the attacker model and target implementation, followed by our experimental setup and fault injection results on the reference implementation of **MAYO-1**. We demonstrate our attack end-to-end with both EMFI and clock glitching.

Attacker model. We consider an attacker with physical access to the board under attack. The attacker needs the same type of board to experimentally find the glitching parameter sets that skip the addition of one, or a few, coefficient(s) in the chosen coordinates of a signature vector. The adversary is assumed to have the capability to trigger executions of the signing procedure on the target device and to inject precisely timed glitches in the clock signal, or electromagnetic pulses with high precision in both time and location on the chip.

Listing 1 C code snippet of the MAYO signature generation, focusing on the addition of oil and vinegar vectors in the MAYO.Sign algorithm.

```
for (int i = 0; i <= k - 1; ++i) {
    vi = Vdec + i * (n - o);
    mat_mul(sk.O, x + i * o, Ox, o, n - o, 1);
    trigger_high();
    mat_add(vi, Ox, s + i * n, n - o, 1);
    memcpy(s + i * n + (n - o), x + i * o, o);
    trigger_low();
}
encode(s, sig, n * k);
```

Target Implementation. We use the **MAYO-1** reference implementation from the pqm4 repository[4], which is a benchmarking framework for post-quantum cryptographic implementations for the ARM Cortex-M4 microcontroller. The C code is compiled using the `arm-none-eabi-gcc` compiler (version 13.2). For the attack, we target the `mat_add` function (as depicted in Listing 1) which performs the addition of the oil and vinegar vectors.

We work with two optimization levels: `-Os` which optimizes for size and `-O3` which is the highest level of optimization targeting execution time. Listing 2 depicts the assembly instructions executed in a given iteration of the `mat_add` function call. Owing to the differences in register allocation and instruction scheduling, we observe a difference in the number of coefficients and relative sequence of instructions. At optimization level `-Os`, the addition operation is performed

[4] `https://github.com/mupq/pqm4/tree/master/crypto_sign/mayo1/` commit 92750c0

Listing 2 Assembly code snippet of the addition of oil and vinegar vectors in MAYO signature generation, compiled with optimization level `-Os` and `-O3`.

```
1    # for -Os
2    LAB_08000636:
3      ldrb.w  r5, [r0], #1  ; r5 = *Vdec; Vdec++  (one vinegar coefficient)
4      ldrb.w  r2, [r3], #1  ; r2 = *Ox; Ox++  (one oil coefficient)
5      adds    r4, #1        ; i++   (loop iterator)
6      eors    r5, r2        ; r5 = r5 ^ r2 (oil added to vinegar)
7      cmp     r4, #78
8      strb.w  r5, [r1], #1  ; (storing vinegar + oil)
9      bne     LAB_08000636
10
11   # for -O3
12   LAB_0800a98c:
13     ldr.w   lr, [r3, #0x4]! ; lr = *(r3+=4) (four oil coefficients)
14     ldr.w   r1, [r0], #0x4  ; r1=*r0; r0=r0+4 (four vinegar coefficients)
15     eor.w   r1, r1, lr      ; r1 = r1 ^ lr (oil added to vinegar)
16     str.w   r1, [r12], #0x4 ; (storing vinegar + oil)
17     ldr     r1, [sp, #0x4c]
18     cmp     r3, r1
19     bne     LAB_0800a98c
```

per coefficient, whereas for `-O3`, four consecutive coefficients are grouped into one register.

Assembly instructions at level `-Os`. Lines 3 and 4 load one vinegar and one oil coefficient into registers `r5` and `r2`, respectively, and line 5 updates the loop iterator. Line 6 performs the addition `r5` $\leftarrow$ `r5` $\oplus$ `r2`, writing the result back to `r5`. Consequently, faulting the load of oil or the addition instruction leaves the original vinegar value in `r5`, which is then exposed in the final signature; faulting the load of vinegar instead reveals the oil coefficient directly.

Assembly instructions at level `-O3`. At this optimization level, four coefficients each of oil and vinegar are loaded into registers `lr` and `r1` (lines 13–14), added together in `r1` (line 15), and the result stored to the address in `r12` (line 16). The same fault pattern applies here however revealing four coefficients at a time: faulting line 13 or line 15 exposes vinegar in the signature, while faulting line 14 exposes oil directly.

Fault Targets. Figure 2 illustrates the order of instructions involved in the oil–vinegar addition across both optimization levels, and the tentative time frame to inject glitches. In each case, the register holding the vinegar value serves as the destination of the addition instruction, so the fault outcome remains the same; what differs is the number of coefficients recovered per fault. We next characterize the fault outcomes obtained by targeting each of these instructions, and how they translate into recovered oil or vinegar coefficients.

Table 4: Summary of number of coefficients per register, fault outcomes, and correction requirements for the `-O3` and `-Os` implementations.

Optimization	#Coefficients	Fault outcome	Correction
`-O3`	4	Oil	Not required
	4	Vinegar	Required
`-Os`	1	Oil	Not required
	1	Vinegar	Required

Fault outcomes. For an execution subjected to fault injection, one of the following three outcomes can occur:

1. *normal execution* where the fault does not impact the execution outcome (no change in the final signature);
2. *faulty execution* wherein the injected fault perturbs the execution and produces an incorrect signature. This event can be split further into three categories, based on which value is revealed in the signature (oil, vinegar, or a different value).
3. *reset*, which initializes a system reset to stabilize the target.

These three scenarios are easily distinguished by observing the signature and using the public verification algorithm. Table 4 summarizes the number of coefficients processed per register, the resulting fault outcomes, and whether signature correction is required for the `-O3` and `-Os` implementations.

4.1 Experimental Setup

Clock Glitching Setup. The MAYO implementation is evaluated on a CW308-STM32F303 target mounted on a CW308 UFO board. The target board houses an ARM Cortex-M4 STM32F303RBT6 processor, operating at 7.37MHz. Faults are injected using the CW-Husky platform via clock glitching. Figure 3 shows the experimental setup, comprising the target board (top) and the CW-Husky (bottom). Short glitches are injected into the clock signal via the HS2 pin in the 20-pin connector on the Husky, potentially resulting in instruction skip or change of instruction operands. The 20-pin connector also facilitates serial communication and handles trigger and reset signals.

EMFI Setup. We perform the EMFI evaluation on the MAYO reference implementation running on a Nucleo-144 development board equipped with an STM32L4R5ZI microcontroller, which integrates an ARM Cortex-M4 processor operating at 32 MHz. We deliberately select a general-purpose development platform, rather than a device specifically designed for physical security evaluation, to demonstrate the practicality of the attack in realistic settings.

Figure 4a depicts our experimental setup, consisting of the target board, an electromagnetic fault injection (EMFI) unidirectional transient probe mounted

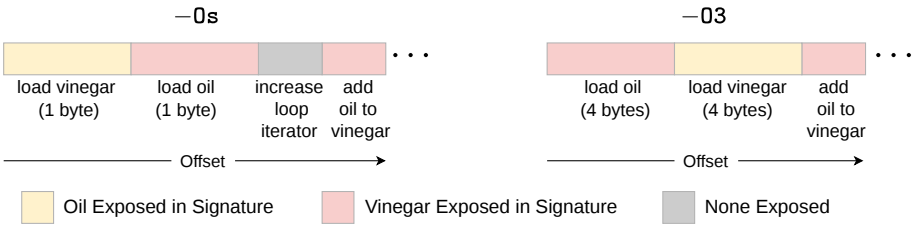


Fig. 2: Instruction ordering and destination register selection in oil–vinegar combination, and their impact on effective glitch offsets

vertically above the chip using an XYZ positioning stage, and a voltage glitching unit (marked in Figure 4a as VC Glitcher). We use the Keysight DS1120A Unidirectional FI Probe [5] with a maximum coil voltage of 450 V (±10%), and maximum current of 64 A. The VC Glitcher is connected to the EM probe via the digital glitch and pulse amplitude connections, and drives the probe coil to generate an electromagnetic pulse. The configurable *glitch source power* controls the energy delivered to the probe. While the target board is powered using an external power supply, the injected EM pulse transiently affects the power supplied to the core. We observe a transient power-rail voltage drop of approximately 2 V lasting for about 50 ns for a given EM injection.

To accurately time the glitch, the target board sends a trigger signal through one of the GPIO pins to the VC Glitcher, which inserts the glitch relative to the trigger signal. The reset pin of VC Glitcher is connected to a MOSFET which power cycles the target before every run, preventing glitches caused by the previous run from affecting the current execution. A host PC orchestrates the experiment by providing inputs to the target, triggering fault injection campaigns, and recording the responses for subsequent analysis. The target board communicates with the host PC over UART to transmit input data and receive corresponding signatures. The VC Glitcher is also connected to the host PC via UART to configure the glitch parameters. We use PicoScope 5203 to monitor the trigger pin, reset pin or the effective coil current transmitted by the probe.

4.2 Fault Injection Attack using Clock Glitching

FI Characterization. The first step is target characterization, which involves scanning a range of glitch width, offset, and external offset parameters, for randomly chosen messages. In clock glitching, the width controls the duration of the glitch, offset controls the position and timing of the glitch with respect to the positive edge of the clock signal, and external offset decides the number of clock cycles that are passed before the glitch is introduced. In CW-Husky, the width and offset parameters are provided in phase shifts. Since the target instruction is

[5] `https://www.keysight.com/us/en/product/DS1120A/unidirectional-faultinjection-probe.html`

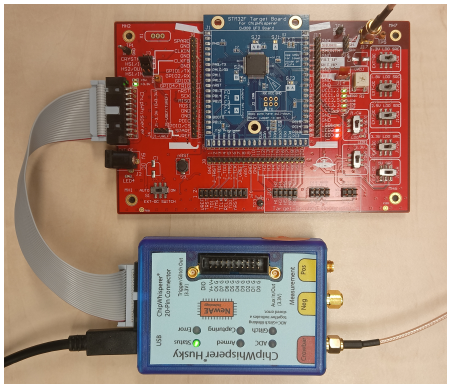
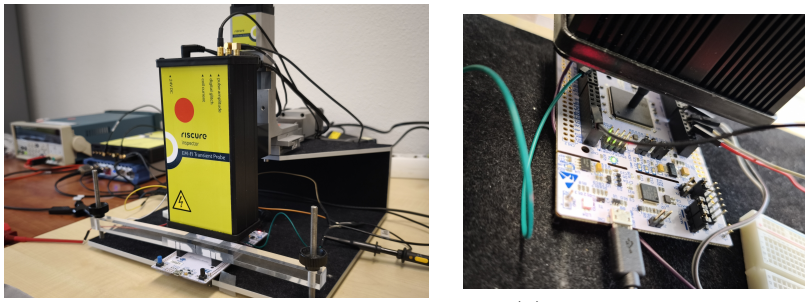


Fig. 3: Experimental Setup for Fault Injection using Clock Glitching. The STM32F3 target (top) executes the MAYO implementation, while the ChipWhisperer Husky (bottom) introduces controlled glitches into the clock signal.



(a) EMFI Setup (b) Probe over Target

Fig. 4: EMFI setup with Nucleo-144 development board as target.

executed in a loop where the coefficients are computed sequentially, the external offset determines which coefficient(s) get impacted.

We performed a search for glitch parameters to identify the parameters that successfully skip either the load instructions or the addition instruction. If the vinegar load is faulted, the oil is exposed in the signature. The oil coefficients can be collected over multiple indices and used for oil-space recovery. On the other hand, faulting the load oil instruction or the addition instruction reveals the vinegar coefficient(s) in the output signature. This realizes the vinegar-revealing attack described in Section 3.1: a successful fault leaves $\mathbf{v}[i]$ rather than $\mathbf{v}[i]+\mathbf{o}[i]$ in the signature. Consequently, the oil coefficient $\mathbf{o}[i]$ is not available directly and has to be recovered using the correction oracle before the pair $(\mathbf{x}, \mathbf{o}[i])$ can be used for oil-space reconstruction. We demonstrate the vinegar-revealing attack in this section.



Fig. 5: Success rate as a function of the target index using clock glitching for `-Os` assembly code.

Characterization results for `-Os`: We found effective glitch widths below 100 and offsets below 100 phase shifts. The external offset for each coefficient is at 9-clock-cycle intervals for the vinegar-revealing fault.
Characterization results for `-O3`: In contrast, the `-O3` implementation processes four coefficients simultaneously. We observed different permutations of vinegar, oil, and random values across the four-byte register. We hypothesize that this behavior results from timing violations extending beyond the target instruction. We could not identify glitch parameters that reveal all four oil or all four vinegar coefficients with sufficiently high probability and therefore restrict the clock-glitching attack to the `-Os` implementation.

Success rate computation. The experiment was done in a realistic scenario, where a different message and salt are fed to the signature algorithm. To determine the success probability of the fault injection, we tested each parameter in the identified parameter set with 20 different inputs to compute how consistently it skipped the addition of the oil coefficient. Figure 5 depicts the success rate for each index computed over 20 inputs that are different than the ones used to identify the glitch parameters. We see that the probability of a successful fault is at least 25%, which is sufficient for a practical attack.

End-to-end Attack Validation. We tested the full attack by generating a pool of $N = 200$ faulted signatures for each targeted coefficient of **MAYO-1** and applying the recovery procedure from Section 3.3. We choose the parameters conservatively based on the lowest observed fault-success probability, $p = 0.25$, for which the expected number of correct signatures in a pool is $Np = 50$.

Following the parameter-selection criteria from Section 3.3, we set the maximum number of sampled subsets to $K = 240\,000$ and accept a candidate row

if

$$\mathrm{agr}(\mathbf{r}) \geq t = 46.$$

Estimating the number of correct signatures by $S \approx Np = 50$, we have

$$M = \binom{200}{8}, \qquad G = \binom{50}{8}.$$

Substituting these values in Equation (6) gives

$$\Pr(\text{at least one good subset in } K \text{ trials}) = 1 - \frac{\binom{M-G}{K}}{\binom{M}{K}} \approx 0.904,$$

which exceeds our target probability of 0.9.

At the same time, we obtain the false-acceptance probability using Equation (8) to be

$$\begin{aligned} &\Pr(\text{at least one incorrect candidate is accepted}) \\ &\quad \leq 240\,000 \cdot \Pr\left(8 + \mathrm{Bin}\left(192, \frac{1}{16}\right) \geq 46\right) \approx 6.03 \cdot 10^{-5}, \end{aligned}$$

which is below the target false-acceptance probability of 10^{-4} per row.

Finally, from Equation (7), at the lowest observed success probability $p = 0.25$,

$$\Pr[\mathrm{agr}(\bar{\mathbf{O}}[i:]) \geq 46] \geq \Pr[\mathrm{Bin}(200, 0.25) \geq 46] \approx 0.767.$$

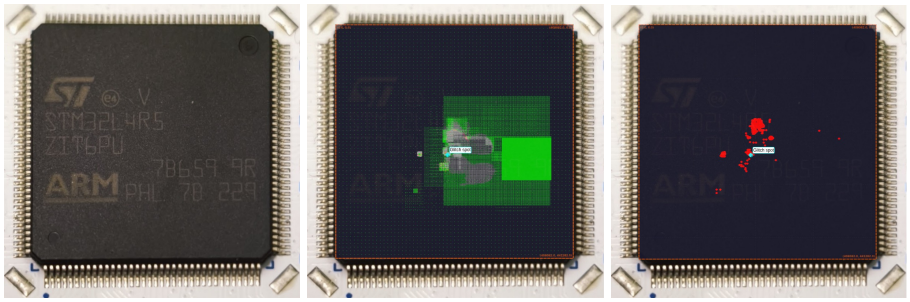
For coefficients with higher fault-success probabilities, this probability increases accordingly. If no candidate reaches the threshold within K trials, we discard the current pool and repeat the procedure with a fresh pool, as described in Section 3.3.

The resulting full key recovery costs are summarized in Table 5. In total, the clock-glitching attack required $15,800$ faulty signatures, 79 pools, and $923,045$ recovery attempts, with an overall recovery time of approximately $198.9\,s$. All 78 rows of the secret oil-space basis were recovered correctly.

4.3 Fault Injection Attack using EMFI

The EMFI attack involves two phases: i) characterization, where an adversary is assumed to have white-box information about the target, and ii) attack where the adversary can observe the final outcome/signature.

Spatial Characterization. The objective of this step is to identify a region of interest for an adversary, where placing the probe will lead to reliable glitches yielding the expected outcomes. Finding an initial configuration that works can be quite challenging, as there is a very large search space if both the (x, y) position and the offset are unknown. To solve this, initially the trigger code is placed immediately around the target operation. In this case, we place the



(a) Chip (b) All results (c) Only Faults

Fig. 6: Merged results of all spatial characterization scans. Here, red indicates a successful fault revealing either a vinegar or an oil value, yellow/gray indicates reset, and green indicates normal execution. The region targeted in the final attack is marked in teal blue.

triggers around a single coefficient addition. This allows the full offset range to be tested quickly when testing an (x, y) position. Once an (x, y) position has been found where the probe can reliably skip the XOR operation, then the trigger window can be widened by setting it around the first call to the `mat_add` function, as shown in Listing 1. Although we found the same spatial position to be effective for both `-Os` and `-O3`, it does not necessarily generalize across different implementations. Figure 6 depicts one of the many scans required to find a region of interest. Each point in this scan consists of 20 runs, where each run executes the addition for a different message. The heatmap highlights the frequency of successful FI when characterizing for optimization level `-Os`. Each point depicts the number of times we obtain a faulty signature. In this scan, we do not yet distinguish between the faults exposing the oil or vinegar values, as either case is interesting for our analysis. From this exercise, we identify the point where we fix our probe position.

Temporal Characterization. We run the FI campaigns for 20 randomized input messages, varying the glitch power and glitch offset. We observe that the optimal glitch source power is 75%, which causes a sharp glitch of -2 V. We set glitch repetition to 1. The target board is operated at 32 MHz, corresponding to a clock cycle of approximately 31.25 ns.

For each coefficient, we identify an offset range where the fault successfully skips the load or addition with high probability, causing the vinegar or oil to be exposed. We vary the offset over the entire duration of the `mat_add` function, injecting an EM pulse every 2 ns. We identify offset windows for each coefficient spanning 50 ns. At the target frequency of 32 MHz, this corresponds to an interval of 1.6 clock cycles. For `-Os`, we identify 78 successful glitch windows, each spanning approximately 50 ns, across the range 100 ns–24,810 ns. For `-O3`, we identify 20 successful windows of approximately 50 ns each within the range 300 ns–5,638 ns.

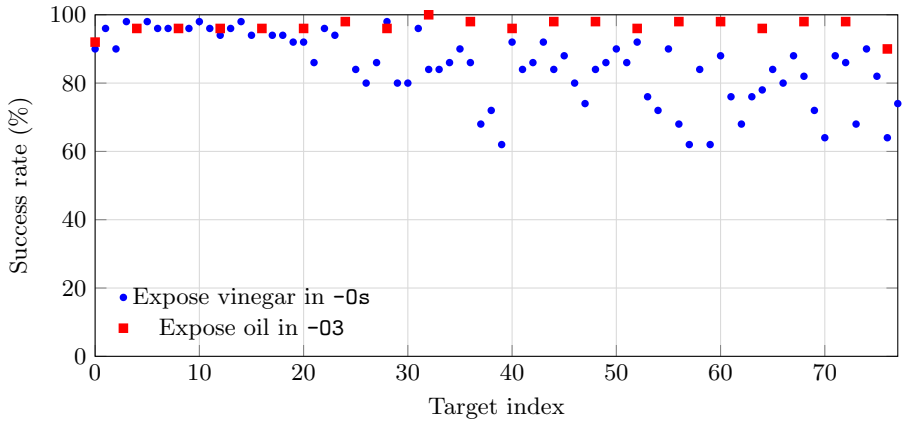


Fig. 7: Success rates as a function of the target index using EMFI. The blue markers show the success rate for exposing the vinegar running `-Os` assembly code, while the red markers show the success rate for exposing the oil running `-O3` code. For the `-O3` experiment, each success rate is assigned to the first target index of the corresponding group.

Figure 2 depicts the order of instructions for `-Os` and `-O3` optimization level, which influences the range of interesting offset values. In particular, we see that for `-Os`, the offset range that exposes vinegar is split in two parts due to the loop increment operation in between the load and XOR operations. In contrast, the assembly snippet for `-O3` provides a continuous range where injecting a glitch will result in the oil being exposed in the signature. Our results validate this behavior, indicating it is more likely to expose the oil values directly for `-O3`, whereas we often see vinegar values being revealed in the signature for `-Os`.

Success rate computation. To measure success rates, we run our attack using the identified parameters on signatures, where we know the secret oil-and-vinegar vectors. Figure 7 compares the success rates of the two recovery strategies as a function of the target index. The success rate for exposing the vinegar in `-Os` is consistently high for the first target indices, mostly between 90% and 98%, but gradually decreases for larger indices. In contrast, exposing the oil in `-O3` is highly reliable for almost all tested groups.

End-to-end Attack Validation. To validate the end-to-end attack using EMFI, we generated a pool of $N = 50$ faulty signatures for each targeted index group and applied the agreement-based oil-space recovery procedure from Section 3.3.

For `-Os`, the exploitable faults expose vinegar coefficients. We therefore first apply the signature-correction procedure to recover the corresponding oil coefficient before using the resulting equation for oil-space reconstruction. For `-O3`, the fault directly exposes the corresponding oil coefficients, so the correction

Table 5: Oil-Space recovery cost for **MAYO-1**. A recovery attempt corresponds to sampling one subset of $o = 8$ signatures. For `-O3`, one pool corresponds to one physical target group covering up to four oil-space rows.

Method	Signatures	Pools	Recovery attempts	Recovery time (s)
Clock glitching	15,800	79	923,045	198.9
EMFI (`-Os`)	3,900	78	1,306	0.198
EMFI (`-O3`)	1,000	20	25	0.017

step is omitted. Apart from this difference, both implementations use the same row-recovery procedure.

The lowest observed exploitable-fault probability in the EMFI experiments is approximately $p = 0.6$. We therefore estimate the number of correct signatures in a pool by $S \approx Np = 30$. Following the parameter-selection criteria from Section 3.3, we set the agreement threshold to $t = 22$ and sample at most $K = 500$ subsets.

With $M = \binom{50}{8}$, $G = \binom{30}{8}$, Eq. (6) gives

$$\Pr(\text{at least one good subset in } K \text{ trials}) = 1 - \frac{\binom{M-G}{K}}{\binom{M}{K}} \approx 0.9958.$$

Moreover, Eq. (7) gives, for the lowest observed success probability,

$$\Pr[\mathrm{agr}(\bar{\mathbf{O}}[i:]) \geq 22] \geq \Pr[\mathrm{Bin}(50, 0.6) \geq 22] \approx 0.9924.$$

Finally, using Equation (8), the false-acceptance probability over at most $K = 500$ sampled subsets is bounded by

$$\begin{aligned} &\Pr(\text{at least one incorrect candidate is accepted}) \\ &\leq 500 \cdot \Pr\left(8 + \mathrm{Bin}\left(42, \frac{1}{16}\right) \geq 22\right) \approx 6.86 \cdot 10^{-5}, \end{aligned}$$

which is below the target bound of 10^{-4} per row.

The resulting end-to-end recovery costs are summarized in Table 5. For `-Os`, recovering the complete oil-space basis required $3,900$ faulty signatures, 78 pools, and $1,306$ recovery attempts, with an overall recovery time of approximately $0.198\,s$. For `-O3`, only $1,000$ faulty signatures, 20 pools, and 25 recovery attempts were required, with an overall recovery time of approximately $0.017\,s$. In both cases, all 78 rows of the secret oil-space basis were recovered correctly.

5 Countermeasures

In this section, we analyze the effectiveness of various countermeasures against our proposed fault injection attack on randomized MAYO and discuss potential adaptations required to bypass them.

Comparison of Signature and Vinegar Vectors. In [24], the authors proposed verifying if the generated signatures differ from the initially produced vinegar vectors, thereby ensuring that oil vectors are properly included. While effective against attacks where the addition of the entire oil vector is omitted, this measure is ineffective against our proposed attack, as our fault model involves skipping only a partial addition of the oil vector. Consequently, signatures will consistently differ from vinegar vectors, making this countermeasure insufficient.

Furthermore, approximately $1/q$ of the coefficients in the chosen oil vectors will be zero during normal execution, resulting in some signature coefficients being equal to the corresponding vinegar coefficients. Extending the check described above to individual coefficients will therefore not work, as correctly generated signatures would almost always be flagged as faulty if no vinegar coefficients were allowed in the signature.

Shuffling. Current implementations of MAYO sequentially perform the addition of vinegar and oil vector components within a deterministic loop structure as in Listing 1. Introducing randomness into this loop, by shuffling the sequence of additions and potentially including randomization in between the components of each vector as well, obscures the exact position targeted by fault injection.

Under shuffling, the attacker additionally has to determine the position affected by the fault. This can be done by exhaustively testing nonzero corrections at all potential coefficients of all signature vectors until a correction is identified through successful verification. The number of additional verifications, therefore, increases by a factor $k(n-o)$, meaning we would require up to $\approx 2^{25}$ verifications for **MAYO-5**. While this increases attack complexity, it remains practically feasible.

Note that circumventing this countermeasure for a similar attack on UOV would have relatively lower overhead, because the signature there consists of only a single vector rather than k as in MAYO.

Sign-Then-Verify. The sign-then-verify countermeasure described in [19] for Dilithium aims to detect signature inconsistencies by performing verification post-generation. In the attack in [19], it is bypassed using exclusively *ineffective faults*.

However, in our case, this strategy would not circumvent this countermeasure for any nonzero row $\bar{\mathbf{O}}[i\ :]$. In order to build the matrix $\mathbf{A}_i$ as in (5) with $a_{i,0} = a_{i,1} = \cdots = a_{i,o-1} = 0$, we would need o linearly independent solutions $\mathbf{x} \in \mathbb{F}_q^o$ to $\bar{\mathbf{O}}[i\ :]\mathbf{x} = 0$.

If such solutions existed, we could construct an invertible $\mathbf{X}_i$ as in Section 3.2, satisfying

$$\bar{\mathbf{O}}[i\ :]\mathbf{X}_i = \mathbf{0}.$$

Since $\mathbf{X}_i$ is invertible, this would imply $\bar{\mathbf{O}}[i\ :] = \mathbf{0}$. Hence, for every nonzero row $\bar{\mathbf{O}}[i\ :]$, it is impossible to obtain the o linearly independent ineffective-fault observations required by our recovery procedure.

This means that the sign-then-verify countermeasure would be effective against our attack. However, verifying after each signature still imposes a substantial

runtime cost, making this approach impractical in performance-sensitive settings.

Masking. Masking, a classical countermeasure against side-channel analysis, can also be used to counteract fault injection attacks. To protect the oil/vinegar combination, every oil and vinegar coefficient can be split into multiple shares. The shares should be recombined carefully to ensure that the recombination and storage operations do not reintroduce an unmasked intermediate value vulnerable to fault injection. Consequently, masking would make our attack much harder, as recovering the secret coefficient would require faulting multiple shares within the same signing execution, which requires multiple precisely placed and reliable faults. This protection comes at the cost of additional computation, and randomness. Increasing the masking order increases the number of shares and hence the implementation overhead, resulting in a security–overhead trade-off.

6 Conclusion and future work

We have presented a novel fault injection attack targeting randomized MAYO signatures, only assuming an attacker can reliably place targeted faults. We demonstrate that the attack can be performed using precise clock glitching and EMFI on ARM Cortex-M4 microcontrollers. By carefully tuning the glitch offset and pulse width parameters, one can introduce faults to skip the step where oil and vinegar coefficients are added. These faults allow the extraction of partial secret information, and by exploiting linear dependencies among the oil vectors, one can efficiently extract the secret key. Notably, the attack does not need a very high success rate of injected faults in order to work.

We also demonstrated that the countermeasure proposed in [24], which checks whether the output signature equals the vinegar vector, is ineffective against our attack. Beyond that, our attack is effective against the countermeasure of randomly shuffling indices, albeit at an increased cost. Among the countermeasures considered, sign-then-verify prevents our attack and masking substantially increases its cost, though both do so at additional implementation cost.

For future work, it would be valuable to study whether the proposed attack can be applied to other multivariate signature schemes, such as SNOVA or QR-UOV.

Acknowledgments We would like to thank Milan Šorf for helpful comments and for pointing out an error in an earlier version of this paper.
This work was supported by the European Union under Grant Agreement No. 101225691 QARC and by the Dutch Research Council (NWO) through the following projects: TTW PREDATOR 19782, BASES 20858, the CiCS project of the research programme Gravitation (024.006.037) and Open competition project ESCAPE (OCENW.XL.23.112).

References

1. Alagic, G., Bros, M., Ciadoux, P., Dang, Q., Dang, T.H., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Silberg, H., Smith-Tone, D., Waller, N.: Status Report on the Second Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process. Internal Report 8610, National Institute of Standards and Technology (May 2026), https://nvlpubs.nist.gov/nistpubs/ir/2026/NIST.IR.8610.pdf
2. Aulbach, T., Campos, F., Krämer, J.: SoK: On the physical security of UOV-based signature schemes. In: Niederhagen, R., Saarinen, M.J.O. (eds.) Post-Quantum Cryptography. pp. 199–231. Springer Nature Switzerland, Cham (2025)
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)
4. Aulbach, T., Marzougui, S., Seifert, J.P., Ulitzsch, V.Q.: MAYo or MAYnot: Exploring implementation security of the post-quantum signature scheme MAYO against physical attacks. In: 2024 Workshop on Fault Detection and Tolerance in Cryptography (FDTC). pp. 28–33 (2024). https://doi.org/10.1109/FDTC64268.2024.00012
5. Bauer, S., Santis, F.D., Koleci, K.: Fault attacks against UOV-based signatures. Cryptology ePrint Archive, Paper 2025/2101 (2025), https://eprint.iacr.org/2025/2101
6. Beullens, W.: MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In: AlTawy, R., Hülsing, A. (eds.) Selected Areas in Cryptography 2021. pp. 355–376. Springer International Publishing, Cham (2021)
7. Beullens, W., Campos, F., Celi, S., Hess, B., Kannwischer, M.J.: Nibbling MAYO: Optimized implementations for AVX2 and Cortex-M4. Cryptology ePrint Archive, Paper 2023/1683 (2023), https://eprint.iacr.org/2023/1683
8. Beullens, W., Chen, M.S., Ding, J., Gong, B., Kannwischer, M.J., Patarin, J., Peng, B.Y., Schmidt, D., Shih, C.J., Tao, C., Yang, B.Y.: UOV. Tech. rep., National Institute of Standards and Technology (2023), https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures
9. 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)
10. Dobias, P., Rezaeezade, A., Chmielewski, Ł., Malina, L., Batina, L.: SoK: Reassessing side-channel vulnerabilities and countermeasures in PQC implementations. Cryptology ePrint Archive, Paper 2025/1222 (2025), https://eprint.iacr.org/2025/1222, AsiaCCS 2026
11. Dummit, D.S., Foote, R.M.: Abstract Algebra, Third Edition. John Wiley and Sons (2004)
12. 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)
13. Gavrilan, I., Oberhansl, F., Wagner, A., Strieder, E., Zankl, A.: Impeccable Keccak: Towards fault resilient SPHINCS+ implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(2), 154–189 (Mar 2024). https://doi.org/10.46586/tches.v2024.i2.154-189

14. 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)
15. Hirner, F., Streibl, M., Krieger, F., Mert, A.C., Roy, S.S.: Whipping the multivariate-based MAYO signature scheme using hardware platforms. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 3421–3435 (2024)
16. Islam, S., Mus, K., Singh, R., Schaumont, P., Sunar, B.: Signature correction attack on Dilithium signature scheme. In: 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P). pp. 647–663 (2022). https://doi.org/10.1109/EuroSP53844.2022.00046
17. Jendral, S., Dubrova, E.: MAYO key recovery by fixing vinegar seeds. IACR Communications in Cryptology **1**(4) (2025). https://doi.org/10.62056/ab0ljbkrz
18. Jendral, S., Dubrova, E., Guo, Q., Johansson, T.: Correction fault attack on CROSS under unknown bit flips. IACR Transactions on Cryptographic Hardware and Embedded Systems **2026**(2), 192–217 (Apr 2026). https://doi.org/10.46586/tches.v2026.i2.192-217
19. Krahmer, E., Pessl, P., Land, G., Güneysu, T.: Correction fault attacks on randomized CRYSTALS-Dilithium. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2024**(3), 174–199 (2024). https://doi.org/10.46586/TCHES.V2024.I3.174-199
20. Krämer, J., Loiero, M.: Fault attacks on UOV and Rainbow. In: Polian, I., Stöttinger, M. (eds.) Constructive Side-Channel Analysis and Secure Design – COSADE 2019. LNCS, vol. 11421, pp. 193–214. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16350-1˙11
21. Mestiri, H., Barraj, I., Machhout, M.: A high-speed KECCAK architecture resistant to fault attacks. In: 2020 32nd International Conference on Microelectronics (ICM). pp. 1–4 (2020). https://doi.org/10.1109/ICM50269.2020.9331792
22. National Institute of Standards and Technology: Module-Lattice-Based digital signature standard. Tech. Rep. NIST FIPS 204, National Institute of Standards and Technology, Gaithersburg, MD (Aug 2024), `https://doi.org/10.6028/NIST.FIPS.204`
23. National Institute of Standards and Technology: Stateless Hash-Based digital signature standard. Tech. Rep. NIST FIPS 205, National Institute of Standards and Technology, Gaithersburg, MD (Aug 2024), `https://doi.org/10.6028/NIST.FIPS.205`
24. 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)
25. 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