SoK: On the Physical Security of UOV-based Signature Schemes

Thomas Aulbach[1], Fabio Campos[2] and Juliane Krämer[1]

[1] University of Regensburg, Germany
{thomas.aulbach,juliane.kraemer}@ur.de
[2] RheinMain University of Applied Sciencess, Wiesbaden, Germany
campos@sopmac.de

Abstract. Multivariate cryptography currently centres mostly around UOV-based signature schemes: All multivariate round 2 candidates in the selection process for additional digital signatures by NIST are either UOV itself or close variations of it: MAYO, QR-UOV, SNOVA, and UOV. Also schemes which have been in the focus of the multivariate research community, but are broken by now - like Rainbow and LUOV - are based on UOV. Both UOV and the schemes based on it have been frequently analyzed regarding their physical security in the course of the NIST process. However, a comprehensive analysis regarding the physical security of UOV-based signature schemes is missing.

In this work, we want to bridge this gap and create a comprehensive overview of physical attacks on UOV and its variants from the second round of NIST's selection process for additional post-quantum signature schemes, which just started. First, we collect all existing side-channel and fault attacks on UOV-based schemes and transfer them to the current UOV specification. Since UOV was subject to significant changes over the past few years, e.g., adaptions to the expanded secret key, some attacks need to be reassessed. Next, we introduce new physical attacks in order to obtain an overview as complete as possible. We then show how all these attacks would translate to MAYO, QR-UOV, and SNOVA. To improve the resistance of UOV-based signature schemes towards physical attacks, we discuss and introduce dedicated countermeasures. As related result, we observe that certain implementation decisions, like key compression techniques and randomization choices, also have a large impact on the physical security, in particular on the effectiveness of the considered fault attacks. Finally, we provide implementations of UOV and MAYO for the ARM Cortex-M4 architecture that feature first-order masking and protection against selected fault attacks. We benchmark the resulting overhead on a NUCLEO-L4R5ZI board and validate our approach by performing a TVLA on original and protected subroutines, yielding significantly smaller t-values for the latter.

Keywords: Multivariate Cryptography · Physical Security · Fault Attacks · Side-channel Analysis · Masking · ARM Cortex-M4 · TVLA

1 Introduction

At the latest since the standards FIPS 203, FIPS 204, and FIPS 205 have been published, post-quantum cryptography can be considered mature enough for practical use. For

*Author list in alphabetical order; see https://www.ams.org/profession/leaders/CultureStatement04.pdf.

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under the project SASPIT (ID 16KIS1858). Furthermore, this work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - project number 50550035. Date of this document: 2024-11-06.

practical applications, however, not only standardized schemes are interesting, but also other post-quantum schemes which offer useful alternatives, for instance, regarding key sizes or computation times. Depending on the application, schemes that are not (yet) standardized might be therefore also in demand.

A very promising post-quantum family for signature schemes with interesting properties is multivariate cryptography. Two multivariate signature schemes, including Rainbow [17], also advanced to the third round of NIST's standardization process for post-quantum cryptography (PQC).[1] However, powerful attacks against both schemes showed that these two schemes should not be standardized [8, 41]. The two attacks, especially the one on Rainbow by Beullens, brought the unbalanced oil and vinegar (UOV) scheme [33] back into the interest of the research community. Although UOV had already been published at the end of the 1990s and is the basis for Rainbow, research concentrated on Rainbow after its publication because it seemed to be more efficient than UOV both in terms of required memory and computation time. This is also why UOV has initially not been submitted to the NIST PQC standardization process. However, although Rainbow is a generalization of the oil-and-vinegar construction underlying UOV, the sweeping attack on Rainbow does not apply to UOV. This makes UOV again a very interesting signature scheme since it withstands cryptanalysis since more than two decades.

UOV in particular and multivariate signature schemes in general feature very small signatures. Short signatures (and fast verification) were also features of signature schemes NIST was explicitly interested in for their call for additional post-quantum signatures.[2] The goal of this process is to diversify the post-quantum signature standards by selecting additional general-purpose signature schemes not based on structured lattices and schemes that are particularly suitable for certain applications. Hence, it was not surprising that ten out of fourty submissions to the call in 2023 have been based on multivariate cryptography.[3] For three of these schemes - 3WISE, DME, and HPPC - rapidly efficient attacks have been found. The remaining seven schemes all rely on the oil-and-vinegar principle, i.e., are UOV-based: MAYO [11], PROV [23], QR-UOV [20], SNOVA [43], TUOV [19], UOV [12], and VOX [34]. Hence, since then, all multivariate signature schemes which are in the focus of the research community are based on the UOV principle. Very recently, on October 25, 2024 NIST announced the 14 schemes to advance to the next round.[4] The share of multivariate signature schemes even increased slightly, since four of the schemes advanced to round 2: MAYO, QR-UOV, SNOVA, and UOV.

When cryptographic schemes are used in practical applications, not only their mathematical security and efficiency, but also their resistance towards physical attacks is important. Hence, post-quantum schemes have been analyzed with respect to their physical security in recent years, and there is also a line of research targeting multivariate cryptography in general and UOV-based signature schemes in particular.

1.1 State of the Art and Related Work

Starting in 2011 [24], UOV-based signature schemes have been analyzed both with respect to passive, i.e., side-channel, and active, i.e., fault attacks. There are results that specifically target UOV [2, 21] or another UOV-based scheme [3, 4, 26, 30, 38, 40], but also publications that analyze several schemes [24, 29, 32].

However, by reading these publications, one does not get a comprehensive picture of the state of the art of the physical security of UOV-based signature schemes. This has

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

[2] https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf

[3] https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures

[4] https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures

several reasons: 1) Some of the schemes have received more attention from the research community than others; simply because of their age, but also because of the existence of a mature implementation. To the best of our knowledge, there was no effort yet to study the transferability of all attacks to all UOV-based signature schemes. 2) Some of the attacks [3, 30] targeted schemes that have since been proven to be insecure, like LUOV and Rainbow. Also for these attacks, it is often unknown if and how they transfer to other schemes. 3) Both the specifications of the schemes and their implementations have been subject to various changes and optimizations over time, e.g., the method of generating compressed public keys introduced in [36]. Hence, it is not even clear if the older attacks still remain a realistic threat to the current version of the algorithms. 4) Although attacks are usually published with descriptions of countermeasures, there are limited results about implemented countermeasures and their overhead, or physically secure implementations of multivariate schemes in general. This might also be due to the fact that for a long time there were no common reference implementations that could serve as a basis. 5) Moreover, since the recent history of multivariate signatures is characterized by major breaks and fixes, e.g., [8, 18, 41], the research community focused initially on the mathematical security of the schemes, which is natural and reasonable.

Now, with NIST's process for standardizing additional digital signature schemes commencing the second round, we have the chance to study all remaining UOV-based schemes on the basis of consistent specifications and implementations. Our goal is to advance their resistance against physical attacks by creating an extensive survey of possible attack vectors, implementing countermeasures, and measuring their overhead directly in comparison to existing implementations.

1.2 Contribution

In this project, we provide a comprehensive overview on the physical security of today's most relevant UOV-based signature schemes, i.e., the schemes MAYO, QR-UOV, SNOVA, and UOV, that are being analyzed in the second round of NIST's standardization process for additional signature schemes. While we analyze all four schemes in this work, we specifically focus on UOV and MAYO, since they provide the more advanced implementation: The more practical a physical attack is carried out, the greater its relevance. To practically perform a physical attack, however, a target implementation is needed. Therefore, UOV and MAYO can be considered the most relevant signature schemes in the field of multivariate cryptography with respect to physical attacks, since they are most advanced in terms of their implementation and both provide an implementation for the ARM Cortex-M4 architecture.

In this work, we provide a complete overview of known side-channel and fault attacks against UOV-based schemes and derive their core attack vectors. We set out to understand if further vulnerabilities exist. In finding new vulnerabilities, we concentrate on side-channel attacks, since so far most attacks are based on faults. We provide a complete overview for all considered schemes regarding their susceptibility towards physical attacks: For all attacks, both known and new, we analyze if and how they can be transferred to all considered schemes. For all attacks, we provide existing and newly developed countermeasures. Moreover, we describe the effect of certain implementation decisions on the physical security of the schemes and derive implementation guidelines from this.

For UOV and MAYO, based on existing optimized M4 implementations, we provide first-order masked implementations for the ARM Cortex-M4 architecture. Additionally, the implementations include protection against the most relevant fault attacks. We benchmark the resulting overhead on a NUCLEO-L4R5ZI board and validate our approach by performing a test vector leakage assessment (TVLA) on original and protected subroutines, yielding significantly smaller t-values for the latter. Our implementation is available to the public at https://github.com/SoK-Psec-UOV-based/code.

1.3 Organization

In Section 2, we discuss the notation used in this work, describe the UOV scheme and the main differences between UOV and MAYO, QR-UOV, and SNOVA, and explain why an attack on a UOV-based scheme often leads to full key recovery once a single oil vector is found. In Sections 3 and 4, we present a comprehensive collection of fault attacks and side-channel attacks, respectively, on UOV-based signature schemes. We present both known and new attacks, and analyze if they can be transferred to the current specifications of UOV, MAYO, QR-UOV, and SNOVA. In Section 5, we describe implementation guidelines that we derived from the analysis of the attacks. In Section 6, we present implementations of UOV and MAYO that include protection against selected fault attacks from Section 3 and countermeasures against all side-channel attacks from Section 4, i.e., are first-order masked. In Section 7, we conclude this work.

2 Background

2.1 Notation

In this paper we describe physical attacks and countermeasures to existing UOV-based signature schemes, which are all submitted to NIST's call for additional digital signatures for the PQC standardization process. Thus, we deem it reasonable to use exactly the notations and conventions introduced in the specification of the designated signature scheme. E.g., the discussions on UOV follow the notation given in [12], the findings about MAYO follow the notation in [11], etc. This requires the reader to be cautious at time, since the respective specifications might use different names or variables for similar objects. Nevertheless, we think this is the correct way, since using a fixed notation in here, would force the reader to adjust the notation on their own when referring to different schemes.

2.2 UOV

Here, we would like to recall UOV in its current form and the most important properties. We will also link its abstract mathematical description to the steps listed in the pseudo code.
The main objects in multivariate cryptography are multivariate quadratic maps $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$. In general, it is hard to find a solution $\mathbf{s} \in \mathbb{F}_q^n$ to a given target $\mathbf{t} \in \mathbb{F}_q^m$ such that $\mathcal{P}(\mathbf{s}) = \mathbf{t}$. This task is also known as the MQ Problem. It can be solved in polynomial time in the very under- or overdetermined case, i.e. $m \geq n(n+1)/2$ or $n \geq m(m+1)$, but is believed to be exponentially hard even for large scale quantum computers if $n \sim m$. By installing a secret trapdoor into the public map $\mathcal{P}$, one can render this task efficiently solvable and construct a signature scheme thereof. In UOV this trapdoor is a m-dimensional linear subspace $O \subset \mathbb{F}_q^n$ - the oil space - with the property $\mathcal{P}(\mathbf{o}) = 0$ for all $\mathbf{o} \in O$. The message μ, together with a random `salt`, is mapped to a target value $\mathbf{t}$ in the codomain $\mathbb{F}_q^m$ using a cryptographic hash function $\mathcal{H}$, i.e $\mathbf{t} = \mathcal{H}(\mu||\texttt{salt})$. Computing a signature boils down to finding a preimage $\mathbf{s} \in \mathbb{F}_q^n$ with $P(\mathbf{s}) = \mathbf{t}$. To this end one can deploy the following method, if knowledge of the oil space is provided. First, one picks a vector $\mathbf{v}$ at random and then solves the equation

$$\mathcal{P}(\mathbf{v} + \mathbf{o}) = \mathcal{P}(\mathbf{v}) + \mathcal{P}(\mathbf{o}) + \mathcal{P}'(\mathbf{v}, \mathbf{o}) = \mathbf{t} \tag{1}$$

for $\mathbf{o} \in O$. The map $\mathcal{P}' : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q^m$ defined by the equation above, is called the differential of $\mathcal{P}$ and is bilinear and symmetric [6]. Viewing the oil vector $\mathbf{o}$ as a linear combination of its m basis vectors given in O ensures $\mathcal{P}(\mathbf{o}) = \mathbf{0}_m$ and shows clearly that

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

$UOV.CompactKeyGen()$ // $csk = (\mathtt{seed}_{\mathtt{sk}}, \mathtt{seed}_{\mathtt{pk}})$

1 : $\mathtt{seed}_{\mathtt{sk}} \leftarrow \{0,1\}^{\mathtt{sk\_seed\_}len}$
2 : $\mathtt{seed}_{\mathtt{pk}} \leftarrow \{0,1\}^{\mathtt{pk\_seed\_}len}$
3 : $\mathbf{O} \leftarrow \mathtt{Expand}_{\mathtt{sk}}(\mathtt{seed}_{\mathtt{sk}})$
4 : $\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}\}_{i\in[m]} \leftarrow \mathtt{Expand}_P(\mathtt{seed}_{\mathtt{pk}})$
5 : **for** $j = i, \ldots, m$ **do**
6 : $\quad \mathbf{P}_i^{(3)} \leftarrow \mathtt{Upper}(-\mathbf{O}^\top \mathbf{P}_i^{(1)} \mathbf{O} - \mathbf{O}^\top \mathbf{P}_i^{(2)})$
7 : $cpk \leftarrow (\mathtt{seed}_{\mathtt{pk}}, \{\mathbf{P}_i^{(3)}\}_{i\in[m]})$
8 : $csk \leftarrow (\mathtt{seed}_{\mathtt{pk}}, \mathtt{seed}_{\mathtt{sk}})$
9 : **return** (cpk, csk)

Figure 1: UOV key generation algorithm

is a system of m linear equations in m variables. If there exists a solution, it can be computed efficiently, if not, one samples a new $\mathbf{v}$ and tries again. Together, the vinegar and oil vector yield a preimage $\mathbf{s} = \mathbf{v} + \mathbf{o}$ to the target $\mathbf{t}$, which constitutes the core of the signature.

2.2.1 Key Generation

Within key generation we need to generate the m-dimensional oil space O and a multivariate quadratic map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ that vanishes on O. The map $\mathcal{P}$ is a sequence of m quadratic polynomials $p_1(\mathbf{x}), \ldots, p_m(\mathbf{x})$ in n variables. The linear and constant part of the polynomials is omitted, and the coefficients of the quadratic monomials can be stored in an upper triangular matrix $\mathbf{P}_i$, such that evaluating the polynomial p_i at a value $\mathbf{a}$ can be realized by computing $p_i(\mathbf{a}) = \mathbf{a}^\top \mathbf{P}_i \mathbf{a}$.

The oil space is chosen to be the space spanned by the rows of a matrix $\bar{\mathbf{O}} = (\mathbf{O}^\top \ \ \mathbf{I}_m)$, where $\mathbf{O} \in \mathbb{F}_q^{(n-m)\times m}$ is sampled uniformly at random. Thus, to guarantee that the quadratic polynomials p_i vanish on O, i.e. $\mathbf{o}^\top \mathbf{P}_i \mathbf{o} = 0$ for all $\mathbf{o} \in O$, one can set the term

$$(\mathbf{O}^\top \ \ \mathbf{I}_m) \begin{pmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ 0 & \mathbf{P}_i^{(3)} \end{pmatrix} \begin{pmatrix} \mathbf{O} \\ \mathbf{I}_m \end{pmatrix} = \mathbf{O}^\top \mathbf{P}_i^{(1)} \mathbf{O} + \mathbf{O}^\top \mathbf{P}_i^{(2)} + \mathbf{P}_i^{(3)}$$

to zero. This is achieved by sampling $\mathbf{P}_i^{(1)} \in \mathbb{F}_q^{(n-m)\times(n-m)}$ (upper triangular) and $\mathbf{P}_i^{(2)} \in \mathbb{F}_q^{(n-m)\times m}$ uniformly at random and setting $\mathbf{P}_i^{(3)}$ accordingly to

$$\mathbf{P}_i^{(3)} = \mathtt{Upper}(-\mathbf{O}^\top \mathbf{P}_i^{(1)} \mathbf{O} - \mathbf{O}^\top \mathbf{P}_i^{(2)}).$$

The described procedure is exactly the key generation algorithm of UOV, which is depicted in Figure 1. Hereby, $\mathbf{O}$ and $\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}$ can be expanded from a private or public seed, respectively.

2.2.2 Secret Key Expansion

As indicated above, it is necessary to solve Equation 2 with respect to $\mathbf{o}$ during signing. The i-th component of $\mathcal{P}'(\mathbf{v}, \mathbf{o})$ is computed by $\mathbf{v}^\top(\mathbf{P}_i + \mathbf{P}_i^\top)\mathbf{o}$ and $\mathbf{o}$ is written as a linear combination of its basis vectors in $\bar{\mathbf{O}}$, where the coefficients $\mathbf{x}$ are the variables we need to solve for. Thus, we can prepare the term $(\mathbf{P}_i + \mathbf{P}_i^\top)\mathbf{o}$ by setting $\mathbf{S}_i = (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top})\mathbf{O} + \mathbf{P}_i^{(2)}$ and store it in the expanded secret key esk, since it is independent from the message μ. The coefficients in $\mathbf{P}_i^{(1)}$ are necessary to compute $\mathcal{P}(\mathbf{v})$, so they are also added to esk. The pseudo code of the secret key expansion is presented in Figure 2.

$UOV.ExpandSK(csk)$ // $csk = (\mathtt{seed}_{\mathtt{sk}}, \mathtt{seed}_{\mathtt{pk}})$

1 : $\mathbf{O} \leftarrow \mathtt{Expand}_{\mathtt{sk}}(\mathtt{seed}_{\mathtt{sk}})$
2 : $\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}\}_{i\in[m]} \leftarrow \mathtt{Expand}_P(\mathtt{seed}_{\mathtt{pk}})$
3 : **for** $j = i, \ldots, m$ **do**
4 : $\quad \mathbf{S}_i \leftarrow (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top})\mathbf{O} + \mathbf{P}_i^{(2)}$
5 : $esk \leftarrow (\mathtt{seed}_{\mathtt{sk}}, \mathbf{O}, \{\mathbf{P}_i^{(1)}, \mathbf{S}_i\}_{i\in[m]})$
6 : **return** esk

Figure 2: Algorithm that expands csk to esk in UOV

$UOV.Sign(esk, \mu)$ // $esk = (\mathtt{seed}_{sk}, \mathbf{O}, \{\mathbf{P}_i^{(1)}, \mathbf{S}_i\}_{i\in[m]})$

1 : $\mathtt{salt} \leftarrow \{0,1\}^{\mathtt{salt\_len}}$
2 : $\mathbf{t} \leftarrow \mathtt{H}(\mu||\mathtt{salt})$ // $\mathbf{t} \in \mathbb{F}_q^m$
3 : **for** $ctr = 0, \ldots, 255$ **do**
4 : $\quad \mathbf{v} \leftarrow \mathtt{Expand}_v(\mu||\mathtt{salt}||\mathtt{seed}_{\mathtt{sk}}||ctr)$ // $\mathbf{v} \in \mathbb{F}_q^{n-m}$
5 : $\quad \mathbf{L} \leftarrow \mathbf{0}_{m\times m}$
6 : $\quad$ **for** $i = 1, \ldots, m$ **do**
7 : $\quad\quad$ Set i-th row of $\mathbf{L}$ to $\mathbf{v}^\top \mathbf{S}_i$
8 : $\quad$ **if** $\mathbf{L}$ is invertible **then**
9 : $\quad\quad \mathbf{y} \leftarrow [\mathbf{v}^\top \mathbf{P}_i^{(1)} \mathbf{v}]_{i\in[m]}$ // $\mathbf{y} \in \mathbb{F}_q^m$
10 : $\quad\quad$ Solve $\mathbf{Lx} = \mathbf{t} - \mathbf{y}$ for $\mathbf{x}$
11 : $\quad\quad \mathbf{s} \leftarrow [\mathbf{v}, \mathbf{0}_m] + \bar{\mathbf{O}}\mathbf{x}$ // $\mathbf{s} \in \mathbb{F}_q^n$
12 : $\quad\quad \sigma \leftarrow (\mathbf{s}, \mathtt{salt})$
13 : $\quad\quad$ **return** σ
14 : **return** 0

Figure 3: Algorithm that signs a message μ in UOV

2.2.3 Signing

The signing algorithm is shown in Figure 3. After generating the `salt` and deriving the target value $\mathbf{t}$, the vinegar vector $\mathbf{v}$ is sampled. Note, that the last m entries of $\mathbf{v}$ are set to zero, which enables a more efficient implementation. With $\mathbf{v}$ at hand, one computes the remaining part of $\mathcal{P}'(\mathbf{v}, \mathbf{o})$ in Line 7, which represents the linear part of the system given in Line 10. In order to evaluate $\mathbf{y} = \mathcal{P}(\mathbf{v})$ in Line 9, only the submatrix $\mathbf{P}_i^{(1)}$ is necessary, since the last entries of $\mathbf{v}$ were chosen to be zero. Solving the derived linear system reveals the coefficients $\mathbf{x}$ of the oil vector $\mathbf{o} = \bar{\mathbf{O}}\mathbf{x} = [\mathbf{Ox}, \mathbf{x}]$. Finally, the sum of the vinegar and oil vector yield the signature $\mathbf{s} = [\mathbf{v}, \mathbf{0}_m] + \bar{\mathbf{O}}\mathbf{x}$.

2.2.4 Public Key Expansion and Verification

To verify that the signer really found a preimage $\mathbf{s}$ under the public map $\mathcal{P}$ of the target vector $\mathbf{t}$, one needs to expand the public coefficients in $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(2)}$ from the seed and check if $\mathcal{P}(\mathbf{s}) = \mathbf{t}$ really holds. We omit the pseudo code of these two functionalities, since we do not focus on them in the main part of this work.

2.2.5 Variants

We have to differentiate between the variants `uov-classic`, `uov-pkc` and `uov-pkc+skc`. In `uov-pkc+skc` the secret key is stored in a compact way *csk*, such that the function *UOV.ExpandSK* is called before signing with *UOV.Sign* and has to be protected as well. In `uov-classic` and `uov-pkc` *UOV.ExpandSK* is part of the key gen, since the secret key is stored in an expanded manner.

One considerable drawback of UOV are its large public keys, due to the amount of coefficients needed to define the public map $\mathcal{P}$. Even in the variants with compressed public keys `uov-pkc+skc` and `uov-pkc`, where a large fraction of the coefficients is expanded from a short seed, the part $\mathbf{P}_i^{(3)}$ still needs to be stored explicitly and contains around $m^3/2$ coefficients in $\mathbb{F}_q$. Here, one factor m comes from the number of polynomials m in the public map $\mathcal{P}$, but the remaining factor $m^2/2$ depends on the dimension of the oil space $dim(O) = m$, which are chosen to be equal in UOV.

2.3 MAYO, QR-UOV and SNOVA

In [7], Beullens presented MAYO, a signature scheme that also employs the oil-and-vinegar principle but reduces the dimension of the oil space from m to o drastically, which in turn reduces the number of coefficients in $\mathbf{P}_i^{(3)}$ to $mo^2/2$ and eventually shrinks down the public key size. The problem is that one can not simply decrease the dimension of the oil space from m to o, with $o \ll m$ and continue as before, since the system $\mathcal{P}'(\mathbf{v}, \mathbf{o}) = \mathbf{t} - \mathcal{P}(\mathbf{v})$ in Equation (2) becomes a system of m linear equations in o variables and bears no solution with high probability. Thus an adaption to the signing and verification procedure was necessary and the following solution was suggested: The public key map $\mathcal{P}$ is stretched into a larger whipped map $\mathcal{P}^* : \mathbb{F}_q^{kn} \to \mathbb{F}_q^m$, such that it accepts k input vectors $\mathbf{s} \in \mathbb{F}_q^n$. This is realized by setting

$$\mathcal{P}^*(\mathbf{s}_1, \ldots, \mathbf{s}_k) \coloneqq \sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}(\mathbf{s}_i) + \sum_{1 \le i < j \le k} \mathbf{E}_{ij}\mathcal{P}'(\mathbf{s}_i, \mathbf{s}_j), \tag{3}$$

where the matrices $\mathbf{E}_{ij} \in \mathbb{F}_q^{m \times m}$ are fixed system parameters with the property that all their non-trivial linear combinations have rank m. The map $\mathcal{P}^*$ vanishes on the subspace $O^k = \{(\mathbf{o}_1, \ldots, \mathbf{o}_k)|$ with $\mathbf{o}_i \in O$ for all $i \in [k]\}$ of dimension ko. This gives back some degrees of freedom when looking for a solution in the new domain $\mathbb{F}_q^{kn}$, i.e. the system obtained from

$$\mathcal{P}^*(\mathbf{v}_1 + \mathbf{o}_1, \ldots, \mathbf{v}_k + \mathbf{o}_k) = \mathbf{t} \tag{4}$$

after randomly sampling and inserting $(\mathbf{v}_1, \ldots, \mathbf{v}_k) \in \mathbb{F}_q^{kn}$, is a system of m linear equations in ko variables. Consequently, if the parameters need are chosen such that $ko > m$, it possible to sample signatures $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_k) = (\mathbf{v}_1 + \mathbf{o}_1, \ldots, \mathbf{v}_k + \mathbf{o}_k)$ similar as before, despite the small dimension of the initial oil space O.

Another attempt in reducing the public key size of UOV is made by QR-UOV. It is very similar to UOV, except that elements of a quotient ring $\mathbb{F}_q[x]/(f)$ are employed, instead of just finite field elements $\mathbb{F}_q$. Let l be a positive integer and $f \in \mathbb{F}_q[x]$. Then any element g of the quotient ring $\mathbb{F}_q[x]/(f)$ uniquely defines a $l \times l$ matrix Φ_g^f over $\mathbb{F}_q$ such that $(1, x, \ldots, x^{l-1}) \cdot \Phi_g^f = (g, xg, \ldots, x^{l-1}g)$. The mapping from g to its polynomial matrix Φ_g^f is an injective ring homomorphism from $\mathbb{F}_q[x]/(f)$ to $\mathbb{F}_q^{l \times l}$ and every element of $\mathcal{A}_f \coloneqq \{\Phi_g^f \in \mathbb{F}_q^{l \times l} | g, \in \mathbb{F}_q[x]/(f)\}$ can be represented by only l elements in $\mathbb{F}_q$. For the construction of QR-UOV, the central maps F_i and public maps P_i are set to be composed of such matrices. This additional structure allows that not every element of these maps are stored since l^2 entries of Φ_g^f can be represented by just l coefficients of g.

SNOVA also work with quotient rings, but apply even more structure to the central and public maps. The central map $F = [F_1, \ldots, F_m] : \mathcal{R}^n \to \mathcal{R}^m$ is defined by

$$F_i = \sum_{\alpha=1}^{l^2} A_\alpha \cdot \left(\sum_{(j,k)\in\Omega} X_j (Q_{\alpha 1} F_{i,jk} Q_{\alpha 1}) X_k \right) \cdot B_\alpha,$$

where the $F_{i,jk}$ are randomly chosen from $\mathcal{R}$, A_α and B_α are invertible elements randomly chosen from $\mathcal{R}$, and $Q_{\alpha 1}, Q_{\alpha 2}$ are invertible matrices randomly chosen from $\mathbb{F}_q[S]$. Here $\mathcal{R}$ is the matrix ring $\mathbb{F}_q^{(l\times l)}$ and $\mathbb{F}_q[S]$ is a commutative symmetric subring of $\mathcal{R}$. Like in UOV, the public matrices are then set to fulfill the equation $P_i = F_i \circ T$ and therefore, are of similar type than the central matrices.
For more details about MAYO, QR-UOV, and SNOVA we refer to the respective specifications given in [11, 20, 43].

2.4 One Vinegar or One Oil Vector Is (In Many Cases) Sufficient for Complete Key Recovery

With respect to signature schemes, it is always an interesting task to determine the amount of data that is necessary to forge signatures, especially from an adversarial point of view. When the secret key is just a seed, as in MAYO, it might become infeasible to recover this seed with side-channel attacks. But it still is useful to obtain information about the values of the expanded seed. Conversely, in `uov-classic` the secret key is of enormous size and only a fraction of it suffices to forge signatures. However, for UOV-based signature schemes, there is a pretty and common answer to this problem: knowledge of the employed oil space O allows to efficiently generate signatures. Even more, there are algebraic methods, that allow to recover the entire space in polynomial time as soon as one or several (depending on the concrete dimensions) oil vectors $\mathbf{o} \in O$ are found.
Regarding MAYO, it was already clear from the description in [7, Section 4.1] that one oil vector is enough to efficiently recover O. The case for UOV was treated in [2] and [35], where again only a single oil vector is needed, since all existing UOV parameter sets fulfill the equation $n - m \leq 2m$.

In [35], Pebereau also elaborated on the very unbalanced case, i.e when $n > 3m$. In this scenario, there might be several oil vectors needed for polynomial time key recovery, namely β of them, where β is the smallest integer such that $n - \beta m \leq 2m$ holds. This becomes interesting, since some parameter sets suggested by QR-UOV and SNOVA lie in the very unbalanced case with $n \geq 6m$ or even up to $n \geq 9m$. In these cases, the security of the scheme might already be compromised when one or two oil vectors are leaked, but attackers are only able to forge signatures in polynomial time, when they get their hands on β of them.

However, if it is possible to reveal one oil vector by means of a physical attack, repeating it on several signing procedures will probably leak more of them. Thus, the attack strategy remains the same, even in the very unbalanced case. Most of the time obtaining an vinegar vector $\mathbf{v}$ is equally strong, since the corresponding oil vector $\mathbf{o}$ can be recovered by subtracting it from the signature $\mathbf{s} = \mathbf{v} + \mathbf{o}$. Consequently, every time one of them is used in a computation, it has to be protected against physical attacks.

3 Fault Attacks on UOV-Based Signatures

In this section, we study the vulnerability of UOV-based signature schemes towards fault attacks. We investigate existing fault attacks on UOV-based signature schemes and provide a more complete catalog of vulnerabilities by also presenting new attacks. Our analysis results in a list of functions that need to be protected in order to achieve an implementation

that is more resistant to fault attacks.
We first analyze the vulnerability of UOV, as this scheme builds the base for all remaining signature schemes analyzed in this work. Subsequently, we explain how these attacks translate to the MAYO signature scheme and its current Cortex M4 implementation. Finally, we consider the case of the remaining UOV-based signatures, QR-UOV and SNOVA. We expect these rather young schemes to be subject to various algorithmical changes and code modifications in the future, thus the findings of this work can be seen as a starting point towards their physical security and as a general estimation whether the respective scheme might be vulnerable to this kind of attack.

To provide a better overview of the existing, transferred, and new vulnerabilities, we summarize the findings of this section in Table 1.

Table 1: Overview of existing and new fault attacks on UOV, MAYO, QR-UOV, and SNOVA. Regarding the feasibility of the attacks, we refer to the specifications submitted to the NIST call for additional signatures in mid-2023. When there are differences between the three UOV-variants `uov-classic`, `uov-pkc` and `uov-pkc+skc`, deterministic and randomized MAYO or the variants `SNOVA-esk` and `SNOVA-ssk`, we list them individually in the given order. With ✓ we state that an attack is possible, while ✗ means the opposite. By $\star$ we denote that an attack is generally possible, but the technical execution is more difficult than in the initially presented attack.

Attack description	Source	Initially for	Feasible in current version	Target
Fix vinegar vector	[3, 4, 24] [26, 29, 40]	Rainbow UOV MAYO	UOV: ✓ MAYO: ✓ QR-UOV: ✓ SNOVA: ✓	`UOV.Sign Line 4` [11] `Alg.8 Line 16,18` [20] `Alg.2 Line 10` [43] `Alg.11 Line 8`
Rowhammer on oil space O	[30]	LUOV	UOV: ✓ | ✓ | $\star$ MAYO: $\star$ QR-UOV: $\star$ SNOVA: ✓ | $\star$	uncompressed secret key in memory
Bit flip on stored secret matrices	[21, 24, 29]	Rainbow UOV	UOV: ✓ | ✓ | $\star$ MAYO: ✓ | ✗ QR-UOV: $\star$ SNOVA: ✗	uncompressed secret key in memory
Prevent addition of oil and vinegar	[38]	MAYO	UOV: ✓ MAYO: ✓ QR-UOV: ✗ SNOVA: ✗	`UOV.Sign Line 11` [11] `Alg.8 Line 45` [20] `Alg.2 Line 18` [43] `Alg.11 Line 14`
Disturb linear system setup	This work	UOV	UOV: ✗ MAYO: ✓ | ✗ QR-UOV: ✗ SNOVA: ✗	`UOV.Sign Line 7` [11] `Alg.8 Line 27,32` [20] `Alg.2 Line 11,12` [43] `Alg.11 Line 9,10`

3.1 Existing Attacks

In the following, we discuss fault attacks against UOV-based schemes that are present in the literature. If the attack was initially not developed for UOV itself, we try to give a detailed analysis of its applicability to UOV.

3.1.1 Fault injection to fix the vinegar vector.

There has been a series of works [3, 4, 26, 29, 40] studying the effectiveness of an instruction skip in Line 4 of Figure 3. These works set the number of necessary faulted signatures to $m = n - v$, which lies in the range from several dozens to hundreds, depending on the security level. Using the findings in [2], [35] and [28] it is now clear that one faulted signature is enough for efficient key-recovery. This can be obtained by the following observations.

If the instruction skip is applied successfully, it leads to the reusage of the vinegar variables that are still stored in $\mathbf{v}$ from a previous signing process. Denote the unfaulty previous signature by $\mathbf{s}$ and the faulty signature by $\tilde{\mathbf{s}}$. The oil vectors $\mathbf{o}$ and $\tilde{\mathbf{o}}$, which are computed during the respective signature generation and added to the vinegar vector $\mathbf{v}$, are different, but both are elements of the oilspace O. Thus, we have

$$\tilde{\mathbf{s}} - \mathbf{s} = (\mathbf{v} + \tilde{\mathbf{o}}) - (\mathbf{v} - \mathbf{o}) = \tilde{\mathbf{o}} - \mathbf{o} \in O,$$

since O is a linear subspace of $\mathbb{F}_q^n$. With a secret oil vector at hand, full key recovery can be achieved in a matter of seconds.

In [26], the authors present two attacks - an absorption skipping and absorption abort attack - directly on the SHAKE256 function that is used to derive the vinegar vectors. These lead to an predictable output of the sampling and therefore reveal the vinegar vectors $\mathbf{v}_i$. They do not make any assumptions about the memory initialization of the device. Since they present this attack specifically on MAYO, we will resume to this in Section 3.3.

3.1.2 RowHammer to alter a value in O.

In [30], the authors present a Rowhammer attack to cause bit flips in the secret transformation $\mathbf{T}$ in LUOV and show how this can be exploited to recover individual bits of $\mathbf{T}$. Repeated execution of the fault attack leads to partial knowledge of the secret $\mathbf{T}$, one bit for every faulty signature. Once enough key bits of $\mathbf{T}$ are recovered, the attacker can apply algebraic analysis techniques to increase the efficiency of the attack and limit the number of faulty signatures that need to be obtained for a full key recovery.
Briefly summarized, the QuantumHammer attack works as follows. The secret data is stored in the DRAMs in memory cells. There is a certain threshold of the voltage level that determines whether a capacitor represents a binary one or zero. If one manages to activate neighboring rows rapidly, this can cause variations in the voltage level of the victim cells due to induction. When a certain threshold is passed, this results in a bit-flip from 0 to 1 or vice versa. Since the attacker does not know at which entry of the secret transformation $\mathbf{T}$ the bit flip occurred, he needs to apply a bit-tracing algorithm to locate the faulted spot and learn the initial value of the flipped bit. The bit flip in $\mathbf{T}$ causes an error in the last part of the signing algorithm in LUOV, where the (vinegar part of the) signature is finally computed by

$$\begin{pmatrix} s_1 \\ \vdots \\ s_v \end{pmatrix} = \begin{pmatrix} t_{11} & \cdots & t_{1m} \\ \vdots & \ddots & \vdots \\ t_{v1} & \cdots & t_{vm} \end{pmatrix} \times \begin{pmatrix} o_1 \\ \vdots \\ o_m \end{pmatrix} + \begin{pmatrix} v_1 \\ \vdots \\ v_v \end{pmatrix}. \tag{5}$$

Thus, a bit flip of the entry t_{ij} leads to a faulty signature entry $\tilde{s}_i$, where the erroneous entry $\tilde{s}_i$ differs from the correct one s_i by o_j. The bit-tracing algorithm now tries to correct the faulted signature by successively adding the values o_k for $k \in \{1, \ldots, m\}$ to the entries s_l for $l \in \{1, \ldots, v\}$. Once the signature is corrected, the position (i, j) of the induced bit flip is successfully located. This works particularly well, since the entries t_{ij} in LUOV are binary, while the values o_l are elements of a larger field, depending on the parameter

choice, e.g., of $\mathbb{F}_{2^8}$. Therefore, the chances that different bit flips in T, say at position t_{ij} and t_{ik}, cause the same error in s are rather small, since then $o_j = o_k$ needs to hold. For more details, please see [30, Section 3.3].

This attack can be transferred to UOV, since the targeted operation in Equation (5) similarly appears in the signing algorithm of UOV, see Line 11 of Figure 3. Here, the oilspace O takes the role of the linear transformation T and they behave equivalently. Since the second block of $\bar{\mathbf{O}} = (\mathbf{O}, \mathbf{I}_m)$ is the identity, the values in $\mathbf{x} \in \mathbb{F}_q^m$ are visible to any attacker as the last m entries of the signature $\mathbf{s}$, analogically to the vector $\mathbf{o}$ in LUOV.

However, when it comes to the bit-tracing algorithm we need to be more careful, since in UOV the entries $t_{i,j}$ are not binary anymore, but elements of $\mathbb{F}_{2^8}$ as well. Let $s_i = \sum_{l=1}^{m} t_{il} \cdot o_l + v_i$ be the entry of the unfaulted signature vector. Introducing a single bit flip to t_{ij} results in an faulted entry $\tilde{s}_i = \sum_{l=1}^{m} t_{il} \cdot o_l + v_i + f_{ij} \cdot o_j$, with $f_{ij} \in \mathbb{F}_{2^8}$ having hamming weight 1. Now, if we have $f_{ij} \cdot o_j = f_{ik} \cdot o_k$, for two different indices $j \neq k$, this implies a bit flip corresponding to f_{ij} results in the same faulted signature entry $\tilde{s}_i$ as a bit flip corresponding to f_{ik} would. The bit-tracing algorithm is still able to correct the faulted signature, but would not be able to decide if the deviation results from f_{ij} or f_{ik} and thus, could not uniquely determine the position of the introduced bit flip.

3.1.3 Bit flip in matrices of `esk`.

See [21, 24, 29] for this attack. It is shown that changing a single coefficient of $\mathbf{F}_i$ leads to reduction of the UOV instance to a smaller one. Recent UOV implementations, including the NIST submission [12], are not working with the central maps $\mathbf{F}_i$ anymore, but include the public matrices $\mathbf{P}_i^{(1)}$ and auxiliary matrices $\mathbf{S}_i = (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top})\mathbf{O} + \mathbf{P}_i^{(2)}$ in their secret key. However, this is merely a change in notation, since the former blocks $\mathbf{F}_i^{(1)}$ and $\mathbf{F}_i^{(2)}$ of $\mathbf{F}_i$ were defined just like that. The analysis [21, 24, 29] can be applied to the new setting accordingly and a single altered coefficient in $\mathbf{P}_i^{(1)}$ or $\mathbf{S}_i$, leads to faulted signature $\mathbf{s}'$, such that $\mathcal{P}(\mathbf{s}')$ and $\mathbf{t}$ indeed deviate in exactly one entry. This difference than allows conclusions about the used oil space O.

3.1.4 Fault injection to skip the addition of the vinegar and oil parts.

This fault attack targets the addition in Line 11 of Figure 3, where the vinegar and oil part are added to receive the signature $\mathbf{s} = \mathbf{v} + \mathbf{o}$. It highly depends on the chosen implementation, but if an attacker is able to exclude the vinegar part by an instruction skip, this is threatening. If there is a way to avoid the contribution of $\mathbf{v}$, this directly reveals $\mathbf{s} = \mathbf{o}$ in the signature, which enables key recovery. Currently, this is implemented via a `memcpy` call that copies $\mathbf{v}$ to $\mathbf{s}$, so this line needs to be protected.

Remark 1. Sampling the vinegar vector $\mathbf{v}$ is randomized by including the `salt` as input to the expand function in Line 4 of Figure 3. If we would deal with a deterministic approach instead, where the signing procedure generates the same vinegar vector $\mathbf{v}$ and outputs identical signatures when a message is signed twice, an adversary could exploit the faulted signature $\mathbf{s}' = \mathbf{v}$ as follows: Subtracting the un-faulted signature $\mathbf{s}$ of the same message $\mathbf{s} - \mathbf{s}' = \mathbf{v} + \mathbf{O}\mathbf{x} - \mathbf{v} = \mathbf{O}\mathbf{x} = \mathbf{o}$ reveals an oil vector.
A similar strategy that also only works in the deterministic setting, is to disturb the computation of the oil vector during signing. If one is able to enforce the computation of a different solution $\mathbf{x}'$ to the linear system, then this again reveals an oil vector. Subtracting the two signatures $\mathbf{s}' - \mathbf{s} = \mathbf{v} + \mathbf{O}\mathbf{x}' - \mathbf{v} - \mathbf{O}\mathbf{x} = \mathbf{O}(\mathbf{x} - \mathbf{x}')$ would cancel out the identical vinegar values and reveal an non-zero oil vector, since $\mathbf{x}' \neq \mathbf{x}$. A similar strategy is applied by the following fault attack.

3.2 New Attack

In addition to the attacks gathered in the section above, we identified the following spots, where exploits by an adversary are conceivable and countermeasures should be applied.

3.2.1 Fault injection to disturb linear system setup (skip the new assignment of L).

Disturbing the linear system $\mathbf{Lx} = \mathbf{t} - \mathbf{y}$ leads to a different solution $\mathbf{x}'$, which might lead to key recovery in the deterministic setting, as explained in Remark 1. Thus, skipping Line 5 of Figure 3 and some or all of the loop in Line 6-7. Hereby, it is important that not a single row of $\mathbf{L}$ remains all zero due to the fault attack. This would imply that the system is not solvable, which results in a second iteration of the signing loop, where $\mathbf{v}$ and $\mathbf{L}$ are refreshed. However, this strategy is only successful when the signature is deterministic and thus, current implementations are not vulnerable.

3.3 Transferability to MAYO

The functionalities of UOV are a subset of those needed to implement MAYO. Therefore, it seems natural to conclude that the listed fault attacks for UOV easily transfer to MAYO. Although this is more or less true, there are some minutiae to bear in mind. 1) The applicability of attacks that target the secret key in memory depends on the format (compressed vs. uncompressed) of the stored keys. MAYO always uses compressed keys, which makes MAYO resistant towards such attacks, although classic UOV is vulnerable. 2) Some attacks depend on the randomization choice of the scheme which might deviate to UOV. More details on these two aspects can be found in Sections 5.1 and 5.2. 3) While UOV works with only one pair of oil and vinegar vectors, there are k of them in MAYO. Nevertheless, a single known oil vector is enough to perform key recovery in polynomial time. Thus, certain attacks might become technically easier in MAYO, since there are k targets available, while only one of them needs to be recovered.

3.3.1 Fault injection to fix one or more vinegar vectors.

This fault attack can be easily transferred to MAYO. Current implementations clear the vinegar vectors at the end of the signing procedure as a security measure to avoid reusing attacks. However, in MAYO one all-zero vinegar vector will not lead to an unsolvable system in Line 37 of MAYO.Sign. This is due to the fact that multiple matrices $\mathbf{M}_i[j,:] = \mathbf{v}_i^\top \mathbf{L}_j$ contribute to the linear part of the system $\mathbf{A}$. Consequently, having some $\mathbf{v}_i$ set to zero will not necessarily result in a non-invertible system and another iteration of the signing loop. Thus, by inserting an instruction skip or loop abort in Line 16 or Line 18 such that one or more (but not all) vinegar vectors remain zero makes the corresponding oil vector $\mathbf{o}_i$ visible in the signature. Furthermore, the authors of [26] present three attacks on this subroutine, in total. Two of them target the absorption phase of the `shake256` internally and the other one forces the unknown input to be zero. In all cases the attacker can predict the outcome of the sampling process which reveals one or more vinegar vectors. Rightfully predicting randomly sampled intermediate values during the signing phase, by directly attacking the `shake256` function, is disastrous in many cryptographic primitives, as is also shown, e.g., in [25]. Thus, we only consider the latter attack as scheme-specific to MAYO, which should be treated with dedicated countermeasures. Instead of zeroing buffers with sensitive information at the end of signing, overwriting these buffers with random data is suggested in [26]. Countermeasures of the aforementioned attacks are discussed in more detail in Section 5.4.

3.3.2 RowHammer to alter a value in O.

In MAYO the secret key only contains a seed $\texttt{seed}_{sk}$. The MAYO.API.sign algorithm (Algorithm 10 in [11]) employs the two functionalities MAYO.ExpandSK (Algorithm 6 in [11]) and MAYO.Sign (Algorithm 8 in [11]) to derive the expanded secret key `esk` from the seed and sign the message using `esk`. The secret oil space $\mathbf{O}$, which is the target of the original fault attack, is therefore not a part of the compressed secret key `csk` Thus, it is not stored in memory permanently and less accessible for bit-flip attempts. Furthermore, the addressed variable is zeroed at the end of the signing procedure as a security measure. This is not depicted in the pseudo code, but realized in the submitted implementations [11]. Thus we do not deem the attack given in [30] to be technically feasible in this scenario.

However, in case an adversary could manage to insert a bit-flip during signing, while $\mathbf{O}$ is stored as part of `esk` in the memory, a similar description as in Section 3.1 would apply, since the oil space takes an equivalent role as in UOV.

3.3.3 Bit flip in matrices of `esk`.

Let $\mathbf{s}'$ be the faulted signature, that is generated when a single bit-flip is applied to either of the matrices $\{\mathbf{P}_i^{(1)}, \mathbf{S}_i\}$ in the expanded secret key, The crucial part of this fault attack against UOV is that $\mathcal{P}(\mathbf{s})$ and $\mathbf{t}$ only deviate in one entry. In contrast, the whipped public map $\mathcal{P}^*(\mathbf{s}_1, ..., \mathbf{s}_k) = \sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}(\mathbf{s}_i) + \sum_{1\leq i<j\leq k} \mathbf{E}_{ij}\mathcal{P}'(\mathbf{s}_i, \mathbf{s}_j)$ in MAYO is of different structure. The $\binom{k}{2}$-many emulsifier maps $\mathbf{E}_{ij}$ and the accumulation of all the transformed terms ensures that $\mathcal{P}^*(\mathbf{s}')$ and $\mathbf{t}$ deviate in most of the entries, even if only a single bit-flip was applied. We confirmed this statement with simulations, where we manually change a single bit in one of the $\mathbf{P}_i^{(1)}$ or $\mathbf{S}_i$. Thus, the introduced fault attack can not be transferred to MAYO, at least not in a straightforward way.
However, for the deterministic variant the situation is different. The bit flips lead to altered solution vectors $\mathbf{x}' = (\mathbf{x}'_1, \ldots, \mathbf{x}'_k)$. This most likely results in an invalid signature $\mathbf{s}' = (\mathbf{s}'_1, \ldots, \mathbf{s}'_k) = (\mathbf{v}_1 + \mathbf{O}\mathbf{x}'_1, \ldots, \mathbf{v}_k + \mathbf{O}\mathbf{x}'_k)$. But if we compare that to the correct signature $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_k) = (\mathbf{v}_1 + \mathbf{O}\mathbf{x}_1, \ldots, \mathbf{v}_k + \mathbf{O}\mathbf{x}_k)$ of the same message, we see that the vinegar part is identical in the deterministic scenario. Consequently, the term $\mathbf{s}_i - \mathbf{s}'_i = \mathbf{O}\mathbf{x}_i + \mathbf{O}\mathbf{x}'_i$ would reveal an oil vector, since $\mathbf{O}\mathbf{x}'_i \in O$.

3.3.4 Fault injection to skip the addition of the vinegar and oil parts.

In general, this fault attack also applies to MAYO. In the MAYO signing procedure [11, Alg.2, Line 45] the sum of k pairs of vinegar and oil vectors is computed. In the current implementation this is achieved by adding both components into another variable `s` via `mat_add(vi, Ox, s + i * param_n, ..., 1);`. Regarding the fault attack, this approach is more secure than first copying `vi` to `s`, and then adding `Ox` to it, as it is done in UOV. However, before the actual addition happens, the vinegar part gets reassigned via `vi = Vdec + i * (param_n - param_o);`. If this instruction can be skipped and `vi` remains empty or assigned with a certain constant value, then an oil vector can be recovered with the approach given in Remark 1.

3.3.5 Fault injection to disturb linear system setup (skip the new assignment of A).

There are plenty of options to introduce faults during the linear system set up in [11, Alg.2] between Line 22 and 33, which might change the solution vectors $\mathbf{x'}_i$ and therefore also the resulting oil vectors $\mathbf{o'}_i$. In the non-randomized setting this could be exploited, similar to the description given in Remark 1.

3.4 Transferability to QR-UOV and SNOVA

In this section we discuss if and how the fault attacks in Table 1 can be transferred to QR-UOV and SNOVA. Both schemes employ the quotient ring structure, but this does not have a large impact on the mentioned attacks.

3.4.1 Fault injection to fix the vinegar vector.

This attack works analogously for QR-UOV and SNOVA. As described in Section 3.1, the crucial part here is that the signature is composed of an oil and a vinegar vector. Then, the repeated usage of two identical vinegar vectors makes them canceling each other out, when those signatures are subtracted from each other. In QR-UOV and SNOVA this last step of mixing the oil and vinegar terms is represented by applying the secret linear transformation $\mathbf{s} = S^{-1}(y_1, \ldots, y_v, y_{v+1}, \ldots, y_n)$ in [20, Alg.2, Line 18] or $\mathbf{sig} = [T](X_0, \ldots, X_{v-1}, \tilde{X}_0, \ldots, \tilde{X}_{o-1})$ in [43, Alg.11, Line 14], where the first v entries represent the vinegar and the last o entries represent the oil variables. Due to the block matrix structure of S and T, with an identity and zero block in the first column, the vinegar variables contribute unaltered to the signature.

3.4.2 RowHammer to alter a value in S or T.

In general, this attack works as initially described for LUOV or adapted to UOV. However, QR-UOV and the variant SNOVA-ssk use compressed secret keys. Consequently, the linear transformations S and T, which are only part of the expanded secret keys, are not permanently stored in memory at a specific location. Thus, from a technical point of view, the execution of the fault attack becomes way more difficult. The time slot where the fault can be induced successfully is reduced to signing time and furthermore, the resulting bit-flip is not permanent, since the secret key will be expanded again in the next signing procedure. However, regarding SNOVA-esk the attack would work equivalently to the original one on LUOV.

3.4.3 Bit flip in matrices of `esk`.

The transfer of this fault attack is again non-trivial. Considering the notation of QR-UOV and SNOVA, there are now the matrices $F_{i,1}, F_{i,1}$ and $F_i^{11}, F_i^{12}, F_i^{21}$ under attack. It is not the quotient-ring structure that determines if a bit-flip in one of these matrices yields useful information for the attacker, but the structure of the public map. In the verification of QR-UOV, the signature is evaluated with the bare public map $\mathcal{P}$, similar to UOV. Therefore, we again are in the case that $\mathcal{P}(\mathbf{sig}')$ and $\mathbf{t}$ only differ in a single entry, which is exploitable.

SNOVA, in contrast, has the whipping structure of MAYO, as pointed out by Beullens in [9]. In Corollary 2 he states that the SNOVA public map can be written as $\mathcal{P}(\mathbf{U}) = \sum_{j=0}^{l-1}\sum_{k=0}^{l-1} \mathbf{E}_{j,k}\mathcal{B}(\mathbf{u}_j, \mathbf{u}_k)$, where the matrices $\mathbf{E}_{j,k}$ have a block diagonal structure with m identical blocks of size $\mathbb{F}_q^{(l^2 \times l^2)}$ on the diagonal. This goes well with the result of our simulations, where a bit-flip introduced to one of the matrices F_i^{11}, F_i^{12}, or F_i^{21}, caused the vectors $\mathcal{P}(\mathbf{sig})$ and $\mathbf{t}$ to deviate in l^2 entries. Thus, the mentioned fault attack can not be applied to SNOVA, at least not without profound modification.

3.4.4 Fault injection to skip the addition of the vinegar and oil parts.

QR-UOV and SNOVA are both randomized schemes. As a result, the only way to mount this attack is to stop the vinegar variables - and only them - from contributing to the signature (see Remark 1).

We analyzed the submitted reference implementation from QR-UOV and came to the conclusion that this is not possible with a first-order fault attack. The signature is computed via `sig->s[i] = Fql_sub(vineger[i], t);`, where the vinegar part `vineger` is not altered or reassigned beforehand and the oil part is encoded in `t`. Skipping this instruction would detain both parts from appearing in `s`.

In SNOVA the case is a little different. They first copy the vinegar entry to the signature `gf16m_clone(signature...[index], X...[index]);` and afterwards add the oil entry to it. However, since this is done entry-wise, aborting the loop or similar strategies would not be successful, as they also prevent the remaining oil entries from contributing. Thus, both schemes do not seem to be vulnerable regarding that attack. Remarkably, this is not due to the quotient ring structure, but to their current implementation details.

3.4.5 Fault injection to disturb linear system setup.

As concluded in Section 3.2, this attack only works in the deterministic setting. Since both QR-UOV and SNOVA do not expand the vinegar variables from a fixed seed, but generate them randomly, the given attack is not feasible.

4 Side-Channel Attacks on UOV-Based Signatures

In this section, we investigate the security of UOV-based signature schemes in terms of side-channel attacks. Similar to the previous section, we first recall and transfer existing attacks to UOV. With the goal of being as exhaustive as possible, we then consider further potential vulnerabilities. The resulting attacks are subsequently adapted to MAYO, QR-UOV, and SNOVA. Table 2 presents an overview of this section.

4.1 Existing Attacks

The following attacks against UOV or a familiar scheme are present in the literature. Both existing attacks target the *UOV.Sign* routine shown in Figure 3 and focus on subroutines where secret data is multiplied with public values.

4.1.1 Power analysis of the evaluation of the vinegar vector.

The target of this side-channel attack is the computation $\mathbf{y} = [\mathbf{v}^\top \mathbf{P}_i^{(1)} \mathbf{v}]_{i\in[m]}$ given in Line 9 of Figure 3. The vinegar vector $\mathbf{v}$ is multiplied from both sides to m matrices $\mathbf{P}_i^{(1)}$ containing public values. This marks an evident entrance door for side channel attacks via power analysis. In [2] the authors showed how to exploit this vulnerability with a profiling attack, that gets along with only a single attack trace. In the profiling phase, the entries v_i of $\mathbf{v}$ are set by hand to certain known values. Then, the considered function is called and power traces are gathered - labeled with the respective value in v_i as reference. During the attack phase, the power trace of the execution of $\mathbf{y} = [\mathbf{v}^\top \mathbf{P}_i^{(1)} \mathbf{v}]_{i\in[m]}$ with the unknown (and secret) vector $\mathbf{v}$ is recorded and compared to the reference traces. The attack trace is likely to have the highest correlation to the reference trace where the identical value v_i is used. In this manner, the entries of $\mathbf{v}$ are revealed, which in turn exposes a secret oil vector $\mathbf{o}$ and enables complete secret key recovery.

4.1.2 Power analysis of the linear subspace matrix multiplication.

The authors of [32] present a differential power analysis (DPA) on the multiplication of the linear transformation $\mathbf{T}$ with the intermediate vector $\mathbf{x}$, that is the solution to the derived linear system. In our notation here, the transformation $\mathbf{T}$ is replaced by the basis $\bar{\mathbf{O}}$ of the linear subspace O. Therefore, this attack can be seen as a power analysis of

Table 2: Overview of existing and new side-channel attacks on UOV, MAYO, QR-UOV, and SNOVA. Regarding the feasibility of the attacks, we refer to the specifications submitted to the NIST call for additional signatures in mid-2023. When there is a difference between deterministic and randomized MAYO, we list them individually in the given order. With ✓ we state that an attack is possible. By ⋆ we denote that an attack is generally possible, but the technical execution is more difficult than in the initially presented attack.

Description: Power analysis ...	Source	Initially Feasible in for		Target
of vinegar evaluation	[2]	UOV	UOV: ✓	`UOV.Sign Line 9`
			MAYO: ✓	[11] `Alg.8 Line 30`
			QR-UOV: ✓	[20] `Alg.2 Line 12`
			SNOVA: ✓	[43] `Alg.8 Line 3,4`
of secret matrix multiplication	[32]	Rainbow UOV	UOV: ⋆	`UOV.Sign Line 11`
			MAYO: ✓ | ⋆	[11] `Alg.8 Line 44`
			QR-UOV: ⋆	[20] `Alg.2 Line 18`
			SNOVA: ⋆	[43] `Alg.11 Line 14`
of linear system setup	This work	UOV	UOV: ✓	`UOV.Sign Line 7`
			MAYO: ✓	[11] `Alg.8 Line 27,32`
			QR-UOV: ✓	[20] `Alg.2 Line 11`
			SNOVA: ✓	[43] `Alg.9 Line 3,4,14,26`
of secret key expansion	This work	UOV	UOV: ✓	`UOV.ExpandSK Line 4`
			MAYO: ✓	[11] `Alg.6 Line 17`
			QR-UOV: ✓	[20] `Alg.2 Line 7`
			SNOVA: ✓	[43] `Alg.6 Line 6,7`
during key generation	This work	UOV	UOV: ✓	`UOV.CompactKeyGen Line 6`
			MAYO: ✓	[11] `Alg.5 Line 16`
			QR-UOV: ✓	[20] `Alg.1 Line 5`
			SNOVA: ✓	[43] `Alg.5 Line 5`

the matrix vector multiplication in Line 11 of Figure 3. The attack takes advantage of the fact that some entries of the vector $\mathbf{x}$ are part of the signature, since they are not altered by the identity block of $\bar{\mathbf{O}}$, resp. T. In more detail, we have $\bar{\mathbf{O}}\mathbf{x} = [\mathbf{O}\mathbf{x}, \mathbf{x}]$ and $\mathbf{s} = [\mathbf{v}, \mathbf{0}_m] + [\mathbf{O}\mathbf{x}, \mathbf{x}]$. Thus, during the computation of $\mathbf{O}\mathbf{x}$, the secret entries in $\mathbf{O}$ are multiplied with known values and are consequently vulnerable to DPA.

The authors in [32] require a few dozen of repeated computations of $\mathbf{O}\mathbf{x}$ to recover the entries in $\mathbf{O}$ by using correlation coefficients. At the time they performed this attack on Rainbow and UOV, these schemes used a deterministic approach. This allowed the authors to make the valid assumption that $\mathbf{x}$ will not change when the same message is signed repeatedly. In the current randomized implementation, the solution vector $\mathbf{x}$ will change with every signing procedure, since every time a new vinegar vector $\mathbf{v}$ is sampled, leading to a completely different linear system $\mathbf{L}\mathbf{x} = \mathbf{t} - \mathbf{y}$.

Hence, the attack will not work in the presented form and needs to be adapted to the new setting. However, we still believe the considered function $\mathbf{O}\mathbf{x}$ needs to be protected, since sensitive data is multiplied with public values, which could be exploited with more evolved analysis methods, like profiling or machine learning techniques.

4.2 New Attacks

Power analysis attacks are possible on various other spots of the UOV functionalities. In *UOV.ExpandSK* and *UOV.CompactKeyGen* there is a bulk of matrix multiplications that

involve the secret matrix $\mathbf{O}$ and public values stored in $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(2)}$, which is clearly vulnerable. Regarding the algorithm *UOV.Sign*, we additionally identified the following operation where caution is required.

4.2.1 Power analysis of the computation of L.

In Line 7 of Figure 3 the linear part $\mathbf{L}$ of the system of equations in Line 10 is computed. Hereby, the i-th row of $\mathbf{L}$ is given by $\mathbf{v}^\top \mathbf{S}_i$. Both components involved are unknown to an attacker, which makes it harder to mount a successful power analysis attack than in the considered scenarios above, where one component is public. However, the matrices $\mathbf{S}_i$ are part of the expanded secret key, and, consequently, remain constant over various signing procedures with the same secret key. If Hamming weight information about one of the factors and their product is leaked, we have seen that blind side-channel attacks [16, 37] can exploit this and reveal information about the considered factors $\mathbf{v}$ and $\mathbf{S}_i$.

4.2.2 Power analysis of the computation of S_i during secret key expansion.

The multiplication in Line 4 of Figure 2 could be analyzed similarly to the side-channel attack against Line 9 of Figure 3. This is also critical and needs to be countered. In the variant `uov-pkc+skc` compressed secret keys are used and the mentioned procedure is part of the signing process. Regarding the other two variants `uov` and `uov-pkc`, where the $\mathbf{S}_i$ are already part of the key, this functionality is attributed to key generation.

4.2.3 Power analysis of the computation of $P_i^{(3)}$ during key generation.

The same holds for the matrix multiplications during key generation, depicted in Line 6 of Figure 1. If an adversary is able to retrieve side-channel information here, these operations also need to be protected, following the same reasoning.

4.3 Transferability to MAYO, QR-UOV, and SNOVA

The task of theoretically transferring the discussed side-channel attacks to MAYO, QR-UOV, and SNOVA is considerably less complicated than it was for the fault attacks. On one hand this is due to the fact that certain implementation choices, like the utilization of compressed keys, have a smaller impact on the effectiveness of side-channel attacks. On the other hand, also the scheme-specific properties are less critical here, since all four schemes contain the typical UOV-like work flow: Generate the vinegar vector(s), compute the constant and linear part of the system of equations, solve the linear system via Gauss, multiply the solution with the oil space to receive corresponding oil vector(s), and finally add the vinegar and oil part together. Thus, the vulnerable subroutines listed in Table 2 need to be executed some way or the other, and whether the scheme uses elements of the field $\mathbb{F}_{2^4}$ or $\mathbb{F}_{2^8}$ or of a quotient ring $\mathbb{F}_q[x]/(f)$, like QR-UOV and SNOVA, is not decisive for the theoretical applicability of the attack, since they also boil down to multiplications over a huge amount of field elements.

The practical execution of the attack, in contrast, will depend heavily on the chosen implementation. In [10], the MAYO team announced that they will change their specification from a bit- to a nibble-sliced representation for their keys etc. Among other things, they present an efficient implementation of MAYO on the Arm Cortex-M4, where the costly matrix multiplications are based on the Method of the Four Russians. This method deviates from previous approaches and to the best of our knowledge, there are no reported side-channel attacks against such implementations in the literature. To perform such a side-channel analysis could pose some interesting challenges and therefore, provides a charming open research questions.

For the remaining schemes QR-UOV and SNOVA, there are currently no Arm Cortex-M4 implementations available, so a concrete analysis of their side-channel security cannot yet be performed. Nevertheless, we hope Table 2 with the respective code lines can provide a good orientation about the vulnerable functions, which need to be treated with caution.

5 Implementation Guidelines

In this section, we present implementation guidelines and dedicated countermeasures to protect implementations of UOV-based signature schemes against the physical attacks presented in this work.

During our research, we identified several theoretical attack vectors - both existing and new ones - that do not lead to physical attacks in practice since the current UOV and MAYO implementations are not vulnerable against them. However, we realized that the implementation decisions that prevent these attacks do not seem to be motivated by preventing physical attacks, primarily. As an example, the utilization of compressed keys first and foremost serves the purpose of reducing key sizes, but at the same time it prevents fault attacks that alter the secret key in memory (cf. Sections 3.3 and 5.2). Hence, we do not consider it correct to term these parts of the implementations *countermeasures*, in contrast to concrete modifications of implementations with the aim of making the implementations more resistant towards physical attacks. Still, we consider it important to list also these implementation decisions in this section to emphasize their importance for physical attack security, which is why we term this section *implementation guidelines* instead of *countermeasures*.

5.1 Randomized Signatures

From a physical security point of view, it is desirable to utilize a randomized signature generation process. For the considered UOV implementation this is already the case. In Line 1 of Figure 3 the `salt` is generated randomly. This `salt` (among others) contributes as input to the `Hash` and `Expand` functions that are used to derive the target vector $\mathbf{t} \in \mathbb{F}_q^m$ and the vinegar vector $\mathbf{v} \in \mathbb{F}_q^{n-m}$. Thus, if the same message is signed twice, the generated signatures (and most of the intermediate values) are different. In contrast, the considered MAYO implementation offers both options - random and deterministic - that determine how the `salt` is derived in Line 10 of the signing algorithm. If the signature computation is deterministic, this is beneficial for an attacker. Subtracting signatures of identical messages leads to vinegar parts canceling each other out, possibly revealing non-zero oil vectors, if one of the two latter fault attacks in Table 1 is applied correctly. Thus, we recommend the usage of the randomized version to prevent both of these attacks. Furthermore, this helps to mitigate side-channel analysis with the goal of obtaining the sampled vinegar vectors. If the vinegar vectors vary between different signing processes, it will be much harder to apply differential power analysis methods. Nevertheless, we suggest masking as an additional countermeasure, see Section 5.5.

5.2 Compressed Keys

In addition to the obvious advantage of reduced key sizes, the use of compressed keys is also beneficial with respect to physical security. If the secret key only consists of a seed, there are less options to introduce exploitable faults while it is stored in memory. Recently, [21] and [30] showed that bit flips introduced to an uncompressed secret key can lead to serious leakage. Even when the precise spot of occurrence is unknown at first, there are methods to localize the bit flips and use them to achieve full key recovery. Moreover,

[30] practically executed the attack on LUOV, emphasizing its relevance for UOV-based schemes.

Using compressed keys prevents both fault attacks described in this work that target the secret key in memory, namely the Rowhammer attack on the secret matrix $\mathbf{O}$ and the one introducing bit flips on the secret matrices.

5.3 Counter RowHammer Fault Attack

However, there are scenarios where key compression techniques are undesirable, e.g., in order to enable a faster signing process. Here, we introduce a method to prevent the RowHammer fault attack on the secret subspace O for such scenarios. To be precise, it is not the secret subspace O which is stored in memory, but a certain basis of it. Right now this basis is represented in standard form, such that the identity part $\mathbf{I}_{m\times m}$ can be omitted and only the remaining $\mathbf{O} \in M_{m\times(n-m)}(\mathbb{F}_q)$ is stored. This method is memory efficient, but the standard form for a given oil space is unique, hence fixed. This enables the bit tracing algorithm used in the RowHammer attack.

Instead one could compute m random - though linearly independent - vectors of O and store this modified basis instead. During signing, we would load the modified basis from memory, compute its standard form and continue signing like usual. Afterwards we again transform the basis to a random one by building random linear combinations. This way, the explicit form of the secret key, i.e., the basis of the subspace O, would change with every signature generation, while the secret information remains the same.

The resulting overhead is obvious. On one hand, the size of the matrix to be stored increases from $M_{m\times(n-m)}(\mathbb{F}_q)$ to $M_{m\times n}(\mathbb{F}_q)$. At first glance, this seems like a considerable drawback, but the fraction of the expanded secret key that is consumed by $\mathbf{O}$ or its enlarged version is rather small compared to the matrices $\mathbf{S}_i$ that are also part of the secret key. Thus, the expanded key size would only increase from 238 to 240 KB in `uov-Ip`.

On the other hand, the effort to compute the standard form at the beginning of the signature generation and the randomized basis at the end, will increase the signing time slightly.

5.4 Modify Vinegar Variable after Usage

The fault attack that leads to re-using or zeroing (most of) the vinegar variables belongs to the most prominent ones in literature, see, e.g., [3, 24, 29, 40]. It leads to valid signatures, since the actual signing process is unaltered by the fault. Only the vinegar variables are forced to values that are either known by the attacker, or have been used before. The part of signing that computes the actual solution to the equation $\mathcal{P}(\mathbf{v}+\mathbf{o}) = \mathbf{t}$ is executed correctly and therefore finds a correct signature. Thus, unlike many fault attacks, it can not be detected by a validity check.

Instead, one actively needs to ensure that the sampled vinegar variables are sound and vary across consecutive signing procedures. To this end, we suggest the following modification. After the sampled vinegar variables are used in Line 7 and 9 of Figure 3, we add a vector with random values to it, i.e., `v += r`. Since the vinegar vector $\mathbf{v}$ is added to the signature s via `s += v` towards the end of the signing process, the component $\mathbf{r}$ needs to be removed from the derived signature at the end by appending the instruction `s += r`.

Obviously this countermeasure could be circumvented with two additional instruction skips, which would lead to a third-order fault attack altogether. An attacker who is able to introduce three independent faults, however, could probably attack a signature algorithm in a simpler way and is therefore not considered a relevant scenario in this work. Irrespective of this, depending on how the additional random value $\mathbf{r}$ is chosen, this countermeasure can even be circumvented more easily: When $\mathbf{r}$ is designed to be a new variable with randomly sampled values, the assignment of the random value to the

variable $\mathbf{r}$ might be skipped, resulting in $\mathbf{r} = 0$. In this case, the countermeasure would be completely circumvented by a single instruction skip, leading to a second-order fault attack altogether, which can be considered to be realistic [14]. To avoid such second-order fault attack, we instead suggest to not initialize $\mathbf{r}$ with randomly sampled values, but to use already existing intermediate values of the signing procedure and directly add them to $\mathbf{v}$. For instance, we can make use of the unknown entries of the vectors $\mathbf{r} \coloneqq \mathbf{v}^\top \cdot \mathbf{S}_i$ for any $i \in \{1, \ldots, m\}$, that are used to compose the linear system which is solved during signing.

This countermeasure can be seen as an approach to randomize the data stored in $\mathbf{v}$ after its usage, which is also suggested by [26]. Since $\mathbf{v}$ is added to the signature $\mathbf{s}$ subsequently, this furthermore employs the idea of infective computing [22]. The component $\mathbf{r}$ is a secret error, which needs to be removed finally. If an injected fault skips the addition of this error, the output of the algorithm will be incorrect and can not be exploited by an attacker.

5.5 Masking against Power Analysis

The listed side-channel attacks in Section 4 all follow a similar concept, namely the power analysis of certain matrix vector multiplications, that ultimately boil down to field multiplications in $\mathbb{F}_q$. The field is rather small, e.g., $q \in \{2^4, 2^8\}$ in UOV or $q = 2^4$ in MAYO, and 32-bit processors, like the ARM Cortex-M family, treat multiple field elements at once. Even though this hampers Hamming weight analysis of the secret or vulnerable values, it has been shown that DPAs [32] or profiling attacks [2] are possible.

The most common countermeasure to prevent power analysis attacks is masking. Currently, to the best of our knowledge there are no masked implementations of UOV-based signature schemes available. In this work, we bridge this gap by providing a first masked version of UOV and MAYO. The goal is to protect the vulnerable intermediate values, mainly the oil and vinegar vectors, whenever they are used, and thereby protect against the existing and newly developed attacks in Section 4.1 and Section 4.2. Since the majority of the utilized functions in signing (and key generation) is linear, they are straightforward to mask. In the following, we provide an overview of the affected lines in the pseudo code and the measures we implemented to mitigate their vulnerability. Therefore, we refer to the pseudocode of UOV, especially the signing algorithm in Figure 3.

- The original implementation uses `shake256` [1] to generate the vinegar vector $\mathbf{v} \in \mathbb{F}_q^n$. In fact it only samples $n - m$ entries of $\mathbf{v}$, as the last m entries are set to zero. Instead, we propose to use a masked version `masked_shake256` [5] to sample two (additive) shares of these entries, that will be used for computations later on and are combined at the end of the signing procedure.

- *Line 7, compute $\boldsymbol{v}^\top \boldsymbol{S}_i$, the linear part of the system of linear equations:* In this step we compute the coefficients of the linear part of the system of equations that needs to be solved during signing. Hereby, the vinegar vector $\mathbf{v}$ is multiplied with numerous matrices $\mathbf{S}_i$, which are part of the (expanded) secret key and do not change in subsequent signature generations. Each resulting vector represents a row in the matrix $\mathbf{L}$ that constitutes the linear system. The matrices $\mathbf{S}_i$ are defined by $\mathbf{S}_i \leftarrow (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top})\mathbf{O} + \mathbf{P}_i^{(2)}$ (see Line 4 of Figure 2) and contain information about the secret oil space O. Therefore, we split them randomly into two additive shares. Since the function $\mathbf{v}^\top \cdot \mathbf{S}_i$ is linear in both components and the vinegar vector already arrives in two shares, we need to compute it four times, one for each combination of the respective two components.

- *Line 9, compute $\boldsymbol{v}^\top \boldsymbol{P}_i^{(1)} \boldsymbol{v}$, which contributes to the constant part of the system of linear equations:* In [2] the authors perform a profiled side-channel attack against

this operation. Since the values in $\mathbf{P}_i^{(1)}$ are public, they can take those matrices as given in the public key and collect profiling traces of this operation for various known values of $\mathbf{v}$. After the profiling phase has finished, only a single attack trace of this operation with the used vinegar vector v is needed to recover its actual value. Thus, masking the vector $\mathbf{v}$ is not an option, since an attacker could just recover the value of both shares with this kind of single trace attack. Instead, we suggest to mask the values given in $\mathbf{P}_i^{(1)}$. To collect meaningful profiling traces it is crucial that an attacker knows the exact value of $\mathbf{P}_i^{(1)}$. By masking the entries of these matrices, we prohibit this strategy and render the collected profiling traces useless. Consequently, we compute $\mathbf{y}_0 = \mathbf{v}^\top \mathbf{P}_{i,0}^{(1)} \mathbf{v}$ and $\mathbf{y}_1 = \mathbf{v}^\top \mathbf{P}_{i,1}^{(1)} \mathbf{v}$ for the two shares $\mathbf{P}_{i,0}^{(1)}$ and $\mathbf{P}_{i,1}^{(1)}$ and continue with the additive shares $\mathbf{y}_0$ and $\mathbf{y}_1$, which contribute to the constant part of the system of linear equations.

- *Line 10, solve $\boldsymbol{Lx} = \boldsymbol{t} - \boldsymbol{y}$ for $\boldsymbol{x}$:* At this point we already arrive with two shares $\mathbf{y}_0$ and $\mathbf{y}_1$ of $\mathbf{y}$. Since we do not want an attacker to get track of the value $\mathbf{y}$, we suggest to continue with these two shares and compute two solutions $\mathbf{x}_0$ and $\mathbf{x}_1$ of the linear systems $\mathbf{L}\mathbf{x}_0 = \mathbf{t} - \mathbf{y}_0$ and $\mathbf{L}\mathbf{x}_1 = -\mathbf{y}_1$. Their sum $\mathbf{x} = \mathbf{x}_0 + \mathbf{x}_1$ gives us the coefficients of the oil vector, since $\mathbf{L}\mathbf{x} = \mathbf{L}(\mathbf{x}_0 + \mathbf{x}_1) = \mathbf{L}\mathbf{x}_0 + \mathbf{L}\mathbf{x}_1 = \mathbf{t} - \mathbf{y}_0 - \mathbf{y}_1 = \mathbf{t} - \mathbf{y}$ just like in the original implementation.

- *Line 11, add together vinegar and oil vector:* First, we compute two shares $\mathbf{o}_0$ and $\mathbf{o}_1$ of the oil vector. To this end, we split the oil space randomly into two additive shares $\mathbf{O}_0$ and $\mathbf{O}_1$. Now, we can compute $\mathbf{o}_0 = \mathbf{O}_0\mathbf{x}$ and $\mathbf{o}_1 = \mathbf{O}_1\mathbf{x}$. We do not need to work with the shares of $\mathbf{x}$ anymore, since $\mathbf{x}$ itself is leaked as part of the signature to the public anyway. Finally, to receive the signature $\mathbf{s}$, we add up all the shares $\mathbf{s} = \mathbf{v}_0 + \mathbf{v}_1 + \mathbf{o}_0 + \mathbf{o}_1$.

We implemented these countermeasures as described above and measured their overhead. The practical results are presented in Section 6. As one could expect, masking the functions in Line 7 and 9 is responsible for the majority of the total overhead induced by masking UOV. These functions are quite expensive themselves and the vast amount of randombytes that is required to generate the shares of the involved matrices also contributes significantly to the increased number of clock cycles, as detailed in Section 6.

6 Practical Results

This section firstly presents the performance evaluation of our protected implementations for UOV and MAYO. As there is currently no Cortex-M4 implementation of QR-UOV and SNOVA publicly available, the practical part of this work focus on UOV and MAYO. In this section, we further provide experimental results in terms of side-channel resistance. For all evaluation purposes, we use existing, unprotected implementations for NIST security level I of both schemes as a basis. More precisely, our implementations are based on the respective optimized UOV (`ov-Ip-pkc/m4f`) and MAYO (`mayo1/m4f`) Cortex-M4 implementation available within the pqm4 [27] library.

To increase compatibility, all changes were applied within the respective signing function itself, i.e. the signature of the function remains unchanged. Note that our findings (cf. Section 5) and implemented measures can be easily applied to other parameter sets.

6.1 Performance Results

In this section we present some performance figures of our first-order masked implementation of UOV and MAYO.

Table 3: Memory requirements for each implementation. Code, data and BSS size listed are in bytes, stack usage in 2^{10} byte (i.e., KiB).

Scheme	Impl.	Library size			Stack usage		
		Code	Data	BSS	keygen	sign	verify
`ov-Ip-pkc`	m4f	80 006	0	0	15.2	5.1	2.5
	m4f-flash	80 062	0	0	401.6	5.1	2.5
	masked-m4f-flash	213 076	0	0	401.6	264.4	2.5
MAYO_1	m4f	16 513	8	0	72.7	110.8	430.3
	masked-m4f	17 630	8	0	72.7	217.2	430.3

For benchmarking, we target the ST NUCLEO-L4R5ZI board featuring an Arm Cortex-M4F core with 640 KB of RAM and 2 MB of flash memory. All randomness required for masking is generated using the internal hardware random number generator available on that board. We used the `arm-none-eabi-gcc` compiler (version 13.3.1) with the compiler flags `-O3 -std=gnu99 -mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16` for compilation.

For the targeted parameter set (`ov-Ip`) of UOV, the combined size of the expanded secret key and the expanded public key is 516 KB. Due to the 640 KB of RAM, both expanded keys fit into the RAM. However, in order to obtain the required space for masking within the 640 KB of RAM on the ST NUCLEO-L4R5ZI board, we applied the approach [13, 15] of writing the keys to flash memory. Table 3 presents the memory requirements of our protected implementations compared to the existing versions. In the case of UOV, it shows 1) the increase of stack usage during the key generation due to writing keys to the flash memory, 2) the additional stack usage when signing due to our implemented masking measures, and 3) the increase of code size required for masking. Whereas in the case of MAYO, the additional memory requirement is significantly lower, as the size of the RAM is sufficient for masking without writing the keys to the flash memory.

Table 4 compares the performance of protected and unprotected versions of UOV. Thereby, we differentiate between certain subroutines (cf. Section 5) to clarify the cost of each measure. In addition to the masked implementation, we implemented blinding as an alternative protecting method for two suitable and most costly subroutines based on the following approach. The functions $\mathbf{v}^\top \mathbf{S}_i$ and $\mathbf{v}^\top \mathbf{P}_i \mathbf{v}$ are linear with respect to the used matrices, so blinding works in a straightforward way. We multiply them with random values $r_i \in \mathbb{F}_q \backslash \{0\}$ beforehand and nullify its effect by multiplying the result with r_i^{-1}. Note that we use different random values r_i for each of the matrices $\mathbf{S}_i$ and $\mathbf{P}_i$ for $i \in \{1, \ldots, m\}$.

This approach is less powerful than masking every single entry of these matrices, but it still ensures that the values in $\mathbf{S}_i$ do not remain identical over various signing procedures and the values in $\mathbf{P}_i$ are not open to public anymore. Table 4 shows that 1) the masked version is about 5× slower than the unprotected implementation and 2) blinding is in total almost 2× slower than masking.

Table 5 compares the performance of protected and unprotected versions of MAYO. Similar to UOV, we present the performance results of each implemented subroutine. The present figures show that the overhead of masking is in total smaller than 2×. Although we followed the same approach for both schemes, the slowdown for UOV is significantly larger compared to MAYO.

Table 4: Cortex-M4F cycle counts for our protected implementations in comparison to the optimized unprotected implementation of `ov-Ip-pkc`. Note that the implementation with blinding only differs from the masked implementation in two subroutines.

Pseudo code	Subroutine	UOV unprotected	[this work] masking	[this work] with blinding
Line 4	Sample vinegar vectors SHAKE256	13 455	132 363	132 363
Line 7	Linear part of system $L = v^\top \cdot S_i$	1 083 775	6 816 989	12 069 951
Line 9	Constant part of system $y = v^\top \cdot P_i \cdot v$	903 390	3 721 735	8 359 110
Line 10	Solve linear system Solve $Lx = t - y$ for x	435 349	872 866	872 866
Line 11	Add oil and vinegar $s = v + Ox$	26 633	109 454	109 454
CM of Sec. 5.4	Modify vinegar after usage v = v + r	-	768	768
Total cycle counts for signing		2 478 708	11 840 264	21 916 475

6.2 Side-channel evaluation

In this section, we present evaluation results for potential side-channel leakages of an unprotected compared to our masked implementation of UOV. All experiments regarding leakage evaluation were carried out using the ChipWhisperer tool chain [31, 42] in Python (version 3.9.5) and performed on a ChipWhisperer-Lite board with an STM32F405 target board featuring an Arm Cortex-M4 core with 192 KB of RAM and 1 MB of flash memory.

Since all vulnerable subroutines (cf. Table 4), with the exception of SHAKE, boil down to multiplications, we focus our efforts on the most costly function `gfmat_prod` which multiplies a vector $\mathbf{v}$ with matrices $\mathbf{S}_i$. Therefore, our implementation for side-channel evaluation only provides the `gfmat_prod` function and all required subroutines for generating traces.

For leakage evaluation, we applied the commonly used Welch's t-test methodology [39]. More precisely, we used the fixed vs. random (FvR) approach. Thereby, we multiply a random vinegar vector with fixed matrices or random matrices, resp. In this case, these matrices represent part of the (extended) secret key.

As shown in Figure 4a, the unmasked implementation is highly leaking by presenting very high t-values in the range of about $(100, -100)$, confirming the threat induced by the leakages. In contrast, the t-values for the masked implementation depicted in Figure 4b are all in the required range of $(-4.5, 4.5)$.

7 Conclusion

In this work we conducted an extensive literature review of all existing physical attacks on UOV-based signature schemes and identified further attack vectors. Since all analyzed schemes share a large amount of operations that are contributed to the oil-and-vinegar principle, the theoretical idea behind the attacks transfers really well across the schemes, both for side-channel and fault attacks. Even the utilization of the quotient ring structure in QR-UOV and SNOVA had no impact on the transferability. The technical realization,

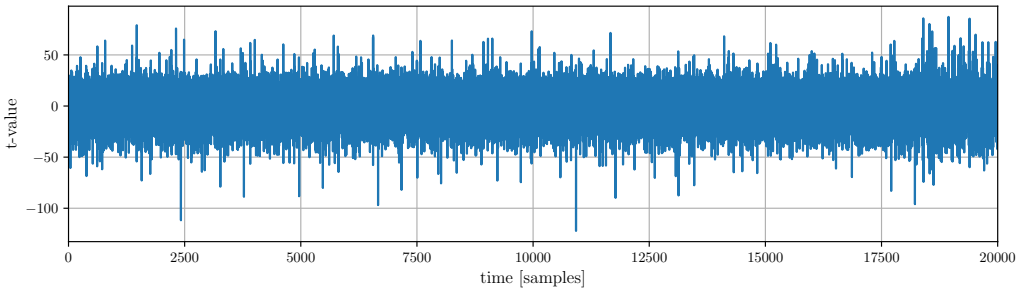
Table 5: Cortex-M4F cycle counts for various subroutines within the expanding and signing procedure in comparison to the unprotected implementation of MAYO_1.

Pseudo code	Subroutine	MAYO unprotected	[this work] masking
[11] `Alg.6 Line 17`	Secret key expansion $L_i = (P_i^{(1)} + P_i^{(1)T})O + P_i^{(2)}$	2 165 338	5 343 609
[11] `Alg.8 Line 16`	Sample vinegar vectors & randomizer SHAKE256	40 775	394 917
[11] `Alg.8 Line 27`	Linear part of system $M_i[j,:] = v_i^T L_j$	524 900	1 782 457
[11] `Alg.8 Line 30`	Constant part of system $u = v_i^T P_a^{(1)} v_i$	1 969 234	3 782 901
[11] `Alg.8 Line 38`	Solve linear system Solve $Ax = y$ for x	928 381	1 858 051
[11] `Alg.8 Line 45`	Add vinegar and oil terms $s_i = (v_i + Ox_i) \mid\mid x_i$	105 209	188 488
Total cycle counts for signing		9 122 185	16 783 809

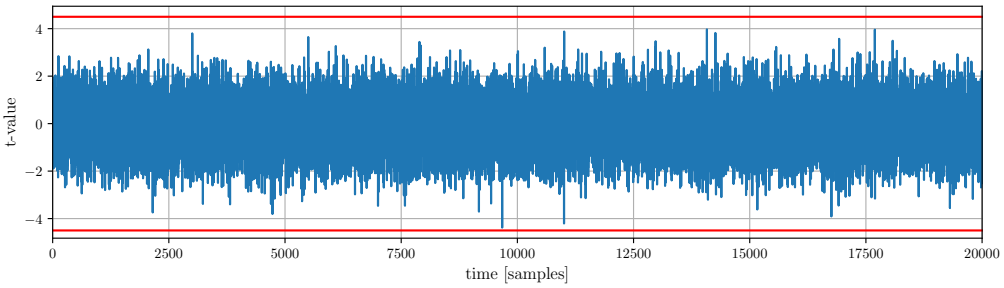
however, depends highly on the given implementation.

We conclude that certain implementation choices, namely the utilization of compressed keys and employing a randomized signing process, has a positive impact on the resistance against fault attacks. The remaining fault attacks can be covered by dedicated countermeasures, with only a small overhead. Note, that we only covered first-order fault attacks in this work.

However, we see a greater risk with regard to side-channel attacks. In every scheme we analyzed, sensitive values are multiplied with huge amounts of public data, which represents a major gateway for power analysis methods. This observation is confirmed by our TVLA conducted on unprotected multiplication routines of UOV. To this end, we present a first-order masked version of both UOV and MAYO on the basis of their existing optimized Cortex-M4 implementations available within the pqm4 library. The results are supposed to serve as a first assessment of the overhead one can expect when applying masking countermeasures to UOV-based schemes. We observed that the produced overhead is smaller for MAYO than for UOV and identified two reasons for that. First, the amount of random bytes that are necessary to split the matrices into two shares is considerably smaller in MAYO, due to smaller parameters. Second, there are some subroutines in MAYO, i.e., the multiplication with public emulsifier maps and the accumulation of these products, where we concluded masking is not required. Thus, the share of sensitive operations is a little higher in UOV than in MAYO.



(a) Unmasked



(b) Masked

Figure 4: Evaluation of the t-test after 10,000 traces for 20 000 samples traced during the computation of the `gfmat_prod` function. The range of the t-values is significantly lower for the masked (b) than for the unmasked version (a). The red lines in (b) indicate the threshold for side-channel leakage.

References

[1] FIPS PUB 202: SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication 202. National Institute of Standards and Technology, U.S. Department of Commerce, 5 2015.

[2] Thomas Aulbach, Fabio Campos, Juliane Krämer, Simona Samardjiska, and Marc Stöttinger. Separating oil and vinegar with a single trace side-channel assisted Kipnis-Shamir attack on UOV. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2023(3):221–245, 2023.

[3] Thomas Aulbach, Tobias Kovats, Juliane Krämer, and Soundes Marzougui. Recovering rainbow's secret key with a first-order fault attack. In Lejla Batina and Joan Daemen, editors, Progress in Cryptology - AFRICACRYPT 2022, pages 348–368, Cham, 2022. Springer Nature Switzerland.

[4] Thomas Aulbach, Soundes Marzougui, Jean-Pierre Seifert, and Vincent Quentin Ulitzsch. 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, 2024.

[5] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Building power analysis resistant implementations of Keccak. Second SHA-3 Candidate Conference, 2010.

[6] Ward Beullens. Improved cryptanalysis of UOV and rainbow. In Anne Canteaut and François-Xavier Standaert, editors, Advances in Cryptology - EUROCRYPT

2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I, volume 12696 of Lecture Notes in Computer Science, pages 348–373. Springer, 2021.

[7] Ward Beullens. MAYO: practical post-quantum signatures from oil-and-vinegar maps. In Riham AlTawy and Andreas Hülsing, editors, Selected Areas in Cryptography - 28th International Conference, SAC 2021, Virtual Event, September 29 - October 1, 2021, Revised Selected Papers, volume 13203 of Lecture Notes in Computer Science, pages 355–376. Springer, 2021.

[8] Ward Beullens. Breaking rainbow takes a weekend on a laptop. In Yevgeniy Dodis and Thomas Shrimpton, editors, Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II, volume 13508 of Lecture Notes in Computer Science, pages 464–479. Springer, 2022.

[9] Ward Beullens. Improved cryptanalysis of SNOVA. IACR Cryptol. ePrint Arch., page 1297, 2024.

[10] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. Nibbling MAYO: optimized implementations for AVX2 and cortex-m4. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2024(2):252–275, 2024.

[11] Ward Beullens, Fabio Campos, Sophía Celi, Basil Hess, and Matthias J. Kannwischer. MAYO. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[12] Ward Beullens, Ming-Shing Chen, Jintai Ding, Boru Gong, Matthias J. Kannwischer, Jacques Patarin, Bo-Yuan Peng, Dieter Schmidt, Cheng-Jhih Shih, Chengdong Tao, and Bo-Yin Yang. UOV. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[13] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. Oil and vinegar: Modern parameters and implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2023(3):321–365, 2023.

[14] Johannes Blömer, Ricardo Gomes da Silva, Peter Günther, Juliane Krämer, and Jean-Pierre Seifert. A practical second-order fault attack against a real-world pairing implementation. In FDTC, pages 123–136. IEEE Computer Society, 2014.

[15] Ming-Shing Chen and Tung Chou. Classic McEliece on the ARM cortex-m4. IACR Trans. Cryptogr. Hardw. Embed. Syst., 2021(3):125–148, 2021.

[16] Christophe Clavier and Léo Reynaud. Improved blind side-channel analysis by exploitation of joint distributions of leakages. In Wieland Fischer and Naofumi Homma, editors, CHES 2017, volume 10529 of LNCS, pages 24–44. Springer, Heidelberg, September 2017.

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

[18] Jintai Ding, Joshua Deaton, Vishakha, and Bo-Yin Yang. The nested subset differential attack - A practical direct attack against LUOV which forges a signature within 210 minutes. In Anne Canteaut and François-Xavier Standaert, editors, EUROCRYPT 2021, Part I, volume 12696 of LNCS, pages 329–347. Springer, Heidelberg, October 2021.

[19] Jintai Ding, Boru Gong, Hao Guo, Xiaoou He, Yi Jin, Yuansheng Pan, Dieter Schmidt, Chengdong Tao, Danli Xie, and Ziyu Yang, Bo-Yin Zhao. TUOV. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[20] Hiroki Furue, Yasuhiko Ikematsu, Fumitaka Hoshino, Tsuyoshi Takagi, Kan Yasuda, Toshiyuki Miyazawa, Tsunekazu Saito, and Akira Nagai. QR-UOV. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[21] Hiroki Furue, Yutaro Kiyomura, Tatsuya Nagasawa, and Tsuyoshi Takagi. A new fault attack on uov multivariate signature scheme. In International Conference on Post-Quantum Cryptography, pages 124–143. Springer, 2022.

[22] Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In Alejandro Hevia and Gregory Neven, editors, LATINCRYPT 2012, volume 7533 of LNCS, pages 305–321. Springer, Heidelberg, October 2012.

[23] Louis Goubin, Benoît Cogliati, Jean-Charles Faugére, Pierre-Alain Fouque, Robin Larrieu, Gilles Macario-Rat, Brice Minaud, and Jacques Patarin. PROV. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[24] Yasufumi Hashimoto, Tsuyoshi Takagi, and Kouichi Sakurai. General fault attacks on multivariate public key cryptosystems. In Bo-Yin Yang, editor, Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, pages 1–18. Springer, Heidelberg, November / December 2011.

[25] Sönke Jendral. A single trace fault injection attack on hedged crystals-dilithium. IACR Cryptol. ePrint Arch., page 238, 2024.

[26] Sönke Jendral and Elena Dubrova. MAYO key recovery by fixing vinegar seeds. IACR Cryptol. ePrint Arch., page 1550, 2024.

[27] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM cortex-m4. IACR Cryptol. ePrint Arch., page 844, 2019.

[28] Namhun Koo and Kyung-Ah Shim. Security analysis of reusing vinegar values in UOV signature scheme. IEEE Access, 12:137412–137417, 2024.

[29] Juliane Krämer and Mirjam Loiero. Fault attacks on UOV and Rainbow. In Ilia Polian and Marc Stöttinger, editors, COSADE 2019, volume 11421 of LNCS, pages 193–214. Springer, Heidelberg, April 2019.

[30] Koksal Mus, Saad Islam, and Berk Sunar. QuantumHammer: A practical hybrid attack on the LUOV signature scheme. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, ACM CCS 2020, pages 1071–1084. ACM Press, November 2020.

[31] Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, Constructive Side-Channel Analysis and Secure Design, pages 243–260, Cham, 2014. Springer International Publishing.

[32] Aesun Park, Kyung-Ah Shim, Namhun Koo, and Dong-Guk Han. 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.

[33] Jacques Patarin. The oil and vinegar signature scheme, 1997.

[34] Jacques Patarin, Benoît Cogliati, Jean-Charles Faugére, Pierre-Alain Fouque, Louis Goubin, Robin Larrieu, Gilles Macario-Rat, and Brice Minaud. VOX. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[35] Pierre Pébereau. One vector to rule them all: Key recovery from one vector in uov schemes. In International Conference on Post-Quantum Cryptography, pages 92–108. Springer, 2024.

[36] Albrecht Petzoldt, Enrico Thomae, Stanislav Bulygin, and Christopher Wolf. Small public keys and fast verification for ultivariate uadratic public key systems. In International Workshop on Cryptographic Hardware and Embedded Systems, pages 475–490. Springer, 2011.

[37] Prasanna Ravi, Dirmanto Jap, Shivam Bhasin, and Anupam Chattopadhyay. Machine learning based blind side-channel attacks on pqc-based kems-a case study of kyber kem. In 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), pages 01–07. IEEE, 2023.

[38] Oussama Sayari, Soundes Marzougui, Thomas Aulbach, Juliane Krämer, and Jean-Pierre Seifert. Hamayo: A fault-tolerant reconfigurable hardware implementation of the MAYO signature scheme. In COSADE, volume 14595 of Lecture Notes in Computer Science, pages 240–259. Springer, 2024.

[39] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings, volume 9293 of Lecture Notes in Computer Science, pages 495–513. Springer, 2015.

[40] Kyung-Ah Shim and Namhun Koo. 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.

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

[42] NewAE Technology. Repository of ChipWhisperer tool chain - commit a9527b5, 2023. https://github.com/newaetech/chipwhisperer.

[43] Lih-Chung Wang, Chun-Yen Chou, Jintai Ding, Yen-Liang Kuan, Ming-Siou Li, Bo-Shu Tseng, Po-En Tseng, and Chia-Chun Wang. SNOVA. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.