Separating Oil and Vinegar with a Single Trace

Side-Channel Assisted Kipnis-Shamir Attack on UOV

Thomas Aulbach[1], Fabio Campos[2,3], Juliane Krämer[1], Simona Samardjiska[3] and Marc Stöttinger[2]

[1] University of Regensburg, Regensburg, Germany
{thomas.aulbach,juliane.kraemer}@ur.de
[2] RheinMain University of Applied Sciences, Wiesbaden, Germany
campos@sopmac.de,marc.stoettinger@hs-rm.de
[3] Radboud University, Nijmegen, Netherlands
simonas@cs.ru.nl

Abstract.
Due to recent cryptanalytical breakthroughs, the multivariate signature schemes that seemed to be most promising in the past years are no longer in the focus of the research community. Hence, the cryptographically mature UOV scheme is of great interest again. Since it has not been part of the NIST process for standardizing post-quantum cryptography so far, it has not been studied intensively for its physical security.
In this work, we present a side-channel attack on the latest implementation of UOV. In the first part of the attack, a single side-channel trace of the signing process is used to learn all vinegar variables used in the computation. Then, we employ a combination of the Kipnis-Shamir attack and the reconciliation attack to reveal the complete secret key. Our attack, unlike previous work, targets the inversion of the central map and not the subsequent linear transformation. It further does not require the attacker to control the message to be signed.
We have verified the practicality of our attack on a ChipWhisperer-Lite board with a 32-bit STM32F3 ARM Cortex-M4 target mounted on a CW308 UFO board. We publicly provide the code and both reference and target traces. Additionally, we discuss several countermeasures that can at least make our attack less efficient.

Keywords: Multivariate signature schemes · UOV · Side-channel attack · Kipnis-Shamir attack · Reconciliation attack

1 Introduction

In July 2022, the National Institute of Standards and Technology (NIST) announced four post-quantum schemes which will soon be standardized, three of which are signature schemes. Two of these signature schemes - Dilithium [DKL+18] and Falcon [PFH+22] - are based on lattices and the third one - SPHINCS$^+$ [BHK+19] - is hash-based. While lattice-based schemes are generally considered to be widely applicable, SPHINCS$^+$ signatures are huge, consuming several KB even for the smallest parameter set. This makes SPHINCS$^+$ useful mainly for specific applications.

Overall, given that currently five families of post-quantum cryptographic assumptions are studied, the three selected signature schemes from only two families imply a lack of diversity. This is problematic for at least two reasons: On the one hand, signature schemes from all five families feature different advantages such as small keys, small signatures, or small computation time. Hence, having standardized signature schemes from more than

two families would allow to choose the optimal scheme for each application. On the other hand, future breakthroughs in (quantum) cryptanalysis might considerably decrease the hardness of post-quantum assumptions, hence decrease the security of the schemes built on them. Having standardized signature schemes from more than two families would provide alternative standardized signature schemes in case one class of assumptions turns out to provide less security than expected, instead of mainly relying on the security of lattices. NIST reacted to this lack of diversity by an explicit call for new signature proposals for the 4th round. For the reasons stated above, NIST is especially interested in signature schemes that are not based on lattices and have small signatures.

Multivariate signature schemes in general feature very small signatures, and the two multivariate signature schemes Rainbow [DCP+20] and GeMSS [CFM+20] also advanced to the third round of NIST's standardization process for post-quantum cryptography (PQC). However, powerful attacks against the third round alternate candidate GeMSS by Tao et al. [TPD21] and the third round finalist Rainbow by Beullens [Beu22a] showed that these two schemes should not be standardized. At this stage, the most relevant signature schemes in the field of multivariate cryptography are MAYO [Beu22b] and UOV [KPG99]. Although UOV has already been published at the end of the 1990s and is the basis for Rainbow, research concentrated on Rainbow after its publication because it is more efficient than UOV both in terms of required memory and computation time. Although Rainbow is a generalization of the oil-and-vinegar construction underlying UOV, Beullen's attack on Rainbow does not apply to UOV. This makes UOV again a very interesting signature scheme since it withstands cryptanalysis since nearly two decades and has very small signatures. Consequently, it will be submitted to the 4th round of NIST's PQC standardization process. Already now, a paper describing modern parameters and implementations of UOV exists [BCH+23a].

Since UOV has initially not been submitted to the NIST PQC standardization process, UOV has also not been in the focus of physical security research until now. Therefore, we set out to analyze the physical security of UOV. In this paper, we propose the first single-trace side-channel attack (SCA) on UOV, targeting the latest UOV implementation [BCH+23a, UOV23].

Related Work Since Rainbow was a third round finalist until 2022, most existing results on the physical security target Rainbow and not UOV. Although both schemes are very similar, not all known results for Rainbow can be transferred to UOV, since UOV does not use the layer structure and the second affine transformation S which mixes the quadratic polynomial equations. Interesting physical attacks have also been proposed against LUOV [BPSV19], an adaptation of the UOV signature scheme that advanced to the second round of NIST's PQC standardization process. We first state results for Rainbow and LUOV that cannot be transferred to UOV and then those which can be transferred to UOV or have been explicitly developed for UOV.

In 2020, Villanueva-Polanco described how to reveal a complete LUOV secret key by a cold boot attack [VP20], i.e., in a setting where an adversary can learn a noisy version of the secret key by cold booting the target device. In the same year, Mus et al. published a hybrid attack on LUOV [MIS20] where they first collect signatures that have been incorrectly computed due to Rowhammer fault injection and then reveal the complete secret key in a divide-and-conquer attack. A correlation power analysis on Rainbow was presented in 2021 by Pokorný et al. [PSN21]. The attack is based on a known message and needs a few hundred power traces to first recover the maps S and T and then reveal the central map F. In 2022, two fault attacks on Rainbow have been presented by Aulbach et al. [AKKM20]. The first attack leads to partial leakage of the secret transformation T by fixing vinegar variables. The second attack induces faults during the application of the linear transformation S. Both attacks eventually lead to full key recovery.

In 2018, Park et al. presented side-channel attacks on Rainbow and UOV [PSKH18]. The attacks are similar to the one described in [PSN21]. They use correlation power analysis together with algebraic key-recovery attacks and demonstrate the practical feasibility of their attack on an 8-bit AVR microcontroller. Again, the attack is based on known messages and first reveals the map S and then the map T. The attack is described for Rainbow but can be transferred to UOV when UOV is implemented with equivalent keys.

Regarding fault attacks not specific to Rainbow, three publications are interesting: Hashimoto et al. described general methods how to attack multivariate cryptography with fault attacks already in 2011 [HTS11]. Based on these ideas, Krämer and Loiero presented two fault attacks on UOV and Rainbow in 2019 [KL19]. In the first attack (to which UOV is immune), a coefficient of the central map F is randomized and in the second attack, vinegar variables are fixed. Just recently, another fault attack on UOV was published [FKNT22], again based on the ideas of [HTS11]. In this attack, a single coefficient of the secret key is faulted.

Contribution In this work, we present the first single-trace side-channel attack against UOV. Our attack targets the latest implementation of UOV [BCH+23a, UOV23] and leads to full key recovery. Contrary to existing work, we target the inversion of the central map during signature generation, not the subsequent linear transformation. Since the target routine of our side-channel attack is the multiplication of (secret) vinegar variables with UOV's secret key, the message to be signed does not need to be controlled nor to be known by the attacker. The attack consists of three steps: First, a single side-channel trace of the inversion of the central map during signature generation allows us to recover the (secret) vinegar variables that are used during the signing process. Therefore, we make use of a correspondence between the public and the secret key, due to the special choice of the linear transformation T. This correspondence exists for both existing versions of UOV - the standard and the compressed version. Then, we use these vinegar variables to recover a vector that is annihilated by the public key, i.e., a vector of the secret linear oil space $\boldsymbol{O}$. In the third step this allows us to apply a modified version of the Kipnis-Shamir attack on reduced parameters that runs in polynomial time and reveals a second oil vector. For all given parameter sets these two oil vectors bear enough information to compute the remaining oil space, i.e., the secret key, by employing an efficient reconciliation attack that only requires us to solve linear equations.

We perform the attack practically on a ChipWhisperer-Lite board with a 32-bit STM32F3 ARM Cortex-M4 target mounted on a CW308 UFO board. The code can be found here: `https://github.com/mstoetti/SCA_assisted_recon_UOV`. We collected reference traces on a profiling device and attack traces on a target device.

Additionally, we provide scripts for collecting reference and target traces on the reader's own ChipWhisperer Setup, if available. The template attack can then be executed, either with the power traces we provide, or with the ones the reader collects. Furthermore, we adapted the Kipnis-Shamir attack [KS06] and the reconciliation attack to accept oil vectors as additional input in order to reduce the complexity. When the side-channel attack recovered an oil vector successfully, the script performs the algebraic attacks to obtain the complete oil space.

We suggest several countermeasures: First we introduce a countermeasure that breaks the correspondence between the public key and the secret key which makes our attack so strong. However, we present a detailed analysis of how the attack can be adapted in that case. Hence, removing this correspondence does not fully prevent the attack, but makes it considerably less powerful. Then we show how known countermeasures such as masking, shuffling, and using precomputations can be applied to UOV.

Organization The rest of this paper is organized as follows: In Section 2, we introduce the UOV algorithm and further background information that is relevant for this work. In Section 3, we present the theoretical part behind our side-channel attack. We present the critical correspondence between the secret key and the public key and show how we can reveal the complete secret UOV key from a single side-channel trace. In Section 4, we present the practical attack. We describe the setup, discuss the parameters we attacked, provide power traces, and discuss how we dealt with noise and what the expected noise resistance of our approach is. Finally, we present countermeasures against our attack in Section 5.

2 Background

In this section, we provide the background knowledge necessary to understand the announced attack. We will present the UOV signature scheme [KPG99], both in its traditional description and the one recently introduced by Beullens [Beu21]. They are equivalent, but facilitate our understanding of different aspects of the attack. Furthermore, we specify details of a major step of the signing process in UOV, the inversion of the central map. Since our attack targets a specific subroutine thereof, it is inevitable to examine this step more closely. The section is concluded by an elaborated presentation of the Kipnis-Shamir attack [KPG99] and the reconciliation attack [DYC+08]. The additional information gained through side-channel leakage, reduces the complexity of the algebraic attacks significantly and allows for an efficient recovery of the complete secret key.

Notation Let $\mathbb{F}_q$ be a finite field with q elements. Let n and m be two positive integers with $n > m$ and $v = n - m$. Subspaces of vector spaces over $\mathbb{F}_q$ are written in bold capital letters, e.g., $\boldsymbol{O} \subseteq \mathbb{F}_q^n$ and their elements have bold letters $\boldsymbol{x}$ with i-th coordinate x_i. Multivariate quadratic maps between those spaces are denoted in calligraphic font $\mathcal{P}$. Their coefficients can be stored in a collection of matrices with capital letters and enumerating superscripts $P^{(k)}$. The matrices we use often have block structure, so we use $P_i^{(k)}$ to denote the submatrices or $0_{m\times v}$ for the zero matrix and I_v for the identity matrix of a certain size.

2.1 Unbalanced Oil and Vinegar Signature Scheme

The essence of UOV consists of a multivariate quadratic map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ that contains a certain elegant trapdoor, namely that $\mathcal{P}$ vanishes on a secret linear subspace $\boldsymbol{O} \subset \mathbb{F}_q^n$ of dimension $\dim(\boldsymbol{O}) = m$. This trapdoor information allows for efficiently obtaining solutions $\boldsymbol{x} \in \mathbb{F}_q^n$ of $\mathcal{P}(\boldsymbol{x}) = \boldsymbol{y} \in \mathbb{F}_q^m$. Without this trapdoor information, finding preimages of $\mathcal{P}$ boils down to solving the multivariate quadratic polynomial (MQ) problem for the system of quadratic equations given by $\mathcal{P}(\boldsymbol{x}) = \boldsymbol{y}$, which is assumed to be hard. There are two common ways to describe how this trapdoor function facilitates the construction of a signature scheme. We will present both of them in the following.

2.1.1 Traditional Description

For around two decades, researchers commonly utilized another multivariate quadratic map $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$, the so-called central map, in order to specify UOV. The polynomials of this map $\mathcal{F} = (f^{(1)}, \ldots, f^{(m)})$ are defined by

$$f^{(k)}(\boldsymbol{x}) = \sum_{1\le i\le n} \sum_{1\le j\le v} \alpha_{i,j}^{(k)} x_i x_j \tag{1}$$

for $k \in \{1, \dots, m\}$, where $\alpha_{i,j}^{(k)} \in \mathbb{F}_q$ represent the coefficients of each quadratic polynomial. Observing the constrains on the indices in Equation (1), this reveals that the linear subspace of dimension m, which consists of vectors with zeros in the first v coordinates, is annihilated by $\mathcal{F}$. We denote this subspace by

$$\boldsymbol{O'} = \{\boldsymbol{x} \mid x_i = 0 \text{ for all } 1 \leq i \leq v\}.$$

The central map $\mathcal{F}$ is concatenated with a random linear transformation $T : \mathbb{F}_q^n \to \mathbb{F}_q^n$, that is supposed to hide the specific structure of $\mathcal{F}$ from anyone who has only access to the concatenation $\mathcal{P} = \mathcal{F} \circ T$. The resulting public key map $\mathcal{P}$ vanishes on the linear subspace $\boldsymbol{O} \coloneqq T^{-1}(\boldsymbol{O'})$. Since all available instantiations of UOV only consider homogeneous polynomials, the coefficients from Equation (1) can be stored in matrices $F^{(k)}$ such that evaluating the polynomial $f^{(k)}$ in $\boldsymbol{x}$ is equivalent to computing $\boldsymbol{x}^\top F^{(k)} \boldsymbol{x}$ for all $k \in \{1, \dots, m\}$. Similarly, we can obtain $n \times n$ matrices $P^{(k)}$ with $p^{(k)}(\boldsymbol{x}) = \boldsymbol{x}^\top P^{(k)} \boldsymbol{x}$ for the public key polynomials $\mathcal{P} = (p^{(1)}, \dots, p^{(m)})$. Thus, from $\mathcal{P} = \mathcal{F} \circ T$, we have

$$P^{(k)} = T^\top F^{(k)} T. \tag{2}$$

By fixing the first v entries $(\tilde{x_1}, \dots, \tilde{x_v})$ - the so-called vinegar variables - in $\boldsymbol{x}$ to certain random elements in $\mathbb{F}_q$ and applying them to Equation (1), we receive polynomials $(\hat{f}^{(1)}, \dots, \hat{f}^{(m)})$. These constitute a linear system of m equations in the remaining m variables $(x_{v+1}, \dots, x_n)$ - the so-called oil variables - of $\boldsymbol{x}$. This system is solvable with quite a high probability and therefore explains why it is possible to find preimages under $\mathcal{F}$.

Key Generation We show a natural method for generating key pairs in Algorithm 1. For any public key map $\mathcal{P}$, there exists an equivalent secret key $(\mathcal{F}, T)$ with

$$T = \begin{pmatrix} I_{v \times v} & T_1 \\ 0_{m \times v} & I_{m \times m} \end{pmatrix}. \tag{3}$$

Thus, in order to obtain a key pair, we first randomly generate T_1. For the central map $\mathcal{F}$ it suffices to randomly generate upper triangular matrices $F^{(k)} \in \mathbb{F}_q^{n \times n}$ since the coefficients $\alpha_{i,j}$ and $\alpha_{j,i}$ can be grouped together. Furthermore, there is a zero block, since we have no quadratic oil terms in Equation (1). Consequently, we only need to generate upper triangular blocks $F_1^{(k)} \in \mathbb{F}_q^{v \times v}$ and blocks $F_2^{(k)} \in \mathbb{F}_q^{v \times m}$ for $k \in \{1, \dots, m\}$. Then, compute $P^{(k)}$ by applying Equation (2) and store the final coefficients again in upper triangular form. There are ways to reduce the key sizes, by, e.g., storing a seed instead of the matrices or storing only T_1 as the secret key. This will be discussed in more detail in Section 3.1.

Signature Generation and Verification Signature generation is displayed in Algorithm 2. To sign a message d, one needs to find a preimage of $\boldsymbol{y} = \mathcal{H}(\mathcal{H}(d)||salt) \in \mathbb{F}_q^m$. This can be done as described above, by turning $\mathcal{F}$ into an invertible linear map $\hat{\mathcal{F}}$. The vinegar and oil variables together represent a solution of $\boldsymbol{x} = \mathcal{F}^{(-1)}(\boldsymbol{y})$. In the next section we will present some algorithmic details of how the described linear system is generated. Finally, we obtain $\boldsymbol{z} \in \mathbb{F}_q^n$ that fulfills $\mathcal{P}(\boldsymbol{z}) = \boldsymbol{y}$ by computing $\boldsymbol{z} = T^{-1}(\boldsymbol{x})$.

Signature Verification boils down to verifying that $\boldsymbol{z}$ is indeed a preimage of $\boldsymbol{y}$ under $\mathcal{P}$.

2.1.2 Beullens' Description

In [Beu21] the author introduces an approach that omits the central map. Here, signing is facilitated directly by knowledge of the secret linear oil space $\boldsymbol{O}$ of dimension m. To this end, consider the polar form of a homogeneous quadratic polynomial defined by

Algorithm 1 UOV Key Generation

Input: Parameters (q, n, v, m)
Output: Key pair (pk, sk)

$T_1 \leftarrow_R \mathbb{F}_q^{v \times m}$
for $k = 1$ `to` m **do**
  $(F_1^{(k)}, F_2^{(k)}) \leftarrow_R (\mathbb{F}_q^{v \cdot (v+1)/2}, \mathbb{F}_q^{v \times m})$
  Build T and $F^{(k)}$ from its blocks, according to Equation (3) and (7)
  Compute $P^{(k)} = T^\top F^{(k)} T$
  $P^{(k)} \leftarrow \texttt{Upper}(P^{(k)})$
end for
$pk \leftarrow \mathcal{P} = (\{P^{(k)}\}_{k=\{1,\dots,m\}})$
$sk \leftarrow (T, \mathcal{F}) = (T, \{F^{(k)}\}_{k=\{1,\dots,m\}})$
return (pk, sk)

Algorithm 2 UOV Signature Generation

Input: message d, private key $(T, \mathcal{F})$, length l of the salt.
Output: signature $\sigma = (\boldsymbol{z}, salt) \in \mathbb{F}_q^n \times \{0,1\}^l$ s.t. $\mathcal{P}(\boldsymbol{z}) = \mathcal{H}(\mathcal{H}(d)||salt)$.

$(x_1, \dots, x_v) \leftarrow_R \mathbb{F}_q^v$
$\hat{F} = \big(\hat{f}^{(1)}, \dots, \hat{f}^{(m)}\big) \leftarrow \big(f^{(1)}(x_1, \dots, x_v), \dots, f^{(m)}(x_1, \dots, x_v)\big)$
if $\operatorname{rank}(\hat{F}) \neq m$ **then**
  return to step 1
end if
$salt \leftarrow_R \{0,1\}^l$
$y \leftarrow \mathcal{H}(\mathcal{H}(d)||salt)$
$x_{v+1}, \dots, x_n \leftarrow \hat{F}^{-1}(y_{v+1}, \dots, y_n)$
$\boldsymbol{z} \leftarrow T^{-1}(\boldsymbol{x})$
$\sigma = (\boldsymbol{z}, salt)$
return σ

$$p'(x, y) \coloneqq p(x + y) - p(x) - p(y) + p(0). \tag{4}$$

The map $p' : \mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q^m$ is a symmetric bilinear form. Assume that P is the matrix associated to the polynomial $p(x)$, then there exists a matrix satisfying $p'(x, y) = x^\top P' y$ and this matrix is given by $P' = P + P^\top$. This notion can be extended to the map $\mathcal{P}$ and we write $\mathcal{P}'$ for the corresponding map.

Given a target $\boldsymbol{t} \in \mathbb{F}_q^m$, we can use $\mathcal{P}'$ to find a preimage under $\mathcal{P}$. Therefore, fix an arbitrary vector $\boldsymbol{v} \in \mathbb{F}_q^n$ and solve the system $\mathcal{P}(\boldsymbol{v} + \boldsymbol{o}) = \boldsymbol{t}$ for a vector $\boldsymbol{o} \in \boldsymbol{O}$. This boils down to solving the following linear system of equations

$$\mathcal{P}(\boldsymbol{v} + \boldsymbol{o}) = \underbrace{\mathcal{P}(\boldsymbol{v})}_{\text{constant}} + \underbrace{\mathcal{P}(\boldsymbol{o})}_{=0} + \underbrace{\mathcal{P}'(\boldsymbol{v}, \boldsymbol{o})}_{\text{linear in } o} = \boldsymbol{t}, \tag{5}$$

because by definition $\mathcal{P}$ vanishes on the secret oil space $\boldsymbol{O}$, i.e., $\mathcal{P}(\boldsymbol{o}) = 0$, for all $\boldsymbol{o} \in \boldsymbol{O}$. Since $\dim(O) = m$, Equation (5) is a linear system of m equations in m variables. If it is solvable, we have found a solution $\boldsymbol{v} + \boldsymbol{o} = \boldsymbol{z} \in \mathbb{F}_q^n$, otherwise, one restarts with a new random vinegar vector $\boldsymbol{v}$.

Remark 1. The characterization of UOV from [Beu21] is an instantiation of what is known as (s, t)-linearity in symmetric key cryptography, first introduced by Boura and Canteaut [BC14] and adapted to multivariate cryptography by Samardjiska and Gligoroski [SG14]. In essence, a function $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ is said to be (s, t)-linear if there exist two linear

subspaces $\boldsymbol{V} \subset \mathbb{F}_q^n$, $\boldsymbol{W} \subset \mathbb{F}_q^m$ with $\dim(\boldsymbol{V}) = s$, $\dim(\boldsymbol{W}) = t$ such that for all $\boldsymbol{w} \in \boldsymbol{W}$, $\boldsymbol{w}^\intercal \cdot f$ has degree at most 1 on all cosets $\boldsymbol{x} + \boldsymbol{V}$ of $\boldsymbol{V}$. It was shown in [SG14] that UOV is (m, m)-linear, i.e., the public map is linear on any coset of the oil space. This is exactly what Equation (5) tells us.

Boura and Canteaut [BC14] further give a characterization of (s, t)–linearity through second order derivatives defined by $D_{\boldsymbol{a},\boldsymbol{b}} f = D_{\boldsymbol{a}} D_{\boldsymbol{b}} f = D_{\boldsymbol{b}} D_{\boldsymbol{a}} f$ where the first order derivative is simply $D_{\boldsymbol{a}} f = f(\boldsymbol{x} + \boldsymbol{a}) - f(\boldsymbol{x})$. Here, f is (s, t)–linear with respect to $\boldsymbol{V}, \boldsymbol{W}$ if and only if all second order derivatives $D_{\boldsymbol{a},\boldsymbol{b}} w^\intercal \cdot f$, with $\boldsymbol{a}, \boldsymbol{b} \in \boldsymbol{V}$, $\boldsymbol{w} \in \boldsymbol{W}$ vanish.

For UOV, this means that $D_{\boldsymbol{o}_1,\boldsymbol{o}_2} \mathcal{P} = 0$ for all oil vectors $\boldsymbol{o}_1, \boldsymbol{o}_2 \in \boldsymbol{O}$. This result was used in [SG14] to provide an alternative description of the reconciliation attack by Ding et al. [DYC+08], that we describe in Section 2.3.2.

Remark 2. We want to point out a difference in notation to prevent potential misunderstanding. In the traditional description, a vector $\boldsymbol{x} \in \mathbb{F}_q^n$ is split by its entries into vinegar and oil variables. In the above description, on the other hand, there are two vectors $\boldsymbol{v}$ and $\boldsymbol{o} \in \mathbb{F}_q^n$, that are called vinegar vector and oil vector, respectively.

Key Generation First, the user generates an oil space $\boldsymbol{O}$ by sampling a uniformly random matrix $O \in \mathbb{F}_q^{v \times m}$, and letting $\boldsymbol{O}$ be the rowspace of (OI_m). Consequently, a multivariate quadratic polynomial $p_k(\boldsymbol{x})$ vanishes on $\boldsymbol{O}$, if for the associated matrix $P^{(k)}$ it holds

$$\begin{pmatrix} O \\ I_m \end{pmatrix}^\top \begin{pmatrix} P_1^{(k)} & P_2^{(k)} \\ 0_{m \times v} & P_3^{(k)} \end{pmatrix} \begin{pmatrix} O \\ I_m \end{pmatrix} = 0. \tag{6}$$

Thus, in order to reduce the key size, we can expand $P_1^{(k)}$ and $P_2^{(k)}$ from a seed and compute $P_3^{(k)}$, such that Equation (6) is fulfilled. This implies the following algorithm for key generation.

Algorithm 3 UOV Key Generation according to [Beu21]

Input: Parameters (q, n, v, m)
Output: Key pair (pk, sk)

1: $O \leftarrow_R \mathbb{F}_q^{v \times m}$
2: $seed \leftarrow_R \{0,1\}^\lambda$
3: **for** $k = 1$ `to` m **do**
4: $\quad (P_1^{(k)}, P_2^{(k)}) \leftarrow$ `Expand`$(seed||k)$
5: $\quad P_3^{(k)} \leftarrow -(OP_1^{(k)}O^\top + OP_2^{(k)})$
6: **end for**
7: $pk \leftarrow (seed, \{P_3^{(k)}\}_{k=\{1,\ldots,m\}})$
8: $sk \leftarrow (seed, O)$
9: **return** (pk, sk)

Signature Generation and Verification The signing process can be directly derived from Equation (5) and is presented in Algorithm 4. Similar to the traditional description, it is possible that the resulting system of linear equations has no solution. In this case, a new vinegar vector has to be generated. Verification is basically done by evaluating $\mathcal{P}(\boldsymbol{z})$, so there is no need for any adaptions.

2.2 Detailed Description of the Central Map Inversion

In Line 2 of Algorithm 2 the randomly generated vinegar variables are inserted into the central map $\mathcal{F}$ to generate a linear system of equations. In the following we give

Algorithm 4 UOV Signature Generation according to [Beu21]

Input: message d, private key $(seed, O)$, length l of the salt.
Output: signature $\sigma = (\boldsymbol{z}, salt) \in \mathbb{F}_q^n \times \{0,1\}^l$ s.t. $\mathcal{P}(\boldsymbol{z}) = \mathcal{H}(\mathcal{H}(d)||salt)$.

1: $salt \leftarrow_R \{0,1\}^l$
2: $\boldsymbol{y} \leftarrow \mathcal{H}(\mathcal{H}(d)||salt)$
3: $\boldsymbol{v} \leftarrow_R \mathbb{F}_q^{n-m} \times \{0\}^m$
4: **if** $\text{rank}(\mathcal{P}(\boldsymbol{v}+\boldsymbol{o})) \neq m$ **then**
5: return to step 3
6: **end if**
7: Solve $\mathcal{P}(\boldsymbol{v}+\boldsymbol{o}) = \boldsymbol{y}$ for $\boldsymbol{o} \in \boldsymbol{O}$
8: $\sigma \leftarrow (\boldsymbol{z} = \boldsymbol{v}+\boldsymbol{o}, salt)$
9: **return** σ

more details on this procedure, as it contains the routine we target in our suggested side-channel attack later on. As described in Section 2.1, the coefficients of $\mathcal{F}$ are stored in the matrices $F^{(k)}$. Due to the structure of the polynomials in Equation (1), i.e., they do not have quadratic oil terms, these matrices are of the form

$$F^{(k)} = \begin{pmatrix} F_1^{(k)} & F_2^{(k)} \\ 0 & 0 \end{pmatrix}. \tag{7}$$

Thus, substituting the vinegar variables $\tilde{\boldsymbol{x}}$ into the central map amounts to computing $\tilde{\boldsymbol{x}}^\top F_1^{(k)} \tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{x}}^\top F_2^{(k)} \boldsymbol{x'}$, where $\boldsymbol{x'} = \{x_{v+1}, \ldots, x_n\}$ are the remaining oil variables of the linear system of equations. Consequently, inserting the vinegar variables into the central map amounts to performing the algebraic operations indicated above for all matrices $F^{(k)}$, where $k \in \{1, \ldots, m\}$.

2.3 Attacks on the Oil & Vinegar Construction

In the following we describe two well-known algebraic attacks on signature schemes that are based on the Oil and Vinegar principle, since we need modified versions of them to complete our side-channel attack.

2.3.1 Kipnis-Shamir Attack

In the original Oil and Vinegar scheme, the two sets of variables were of the same size, i.e., $m = v$ and $n = 2v$. This revealed some weaknesses that were used by Kipnis and Shamir to break the scheme [KS06]. The main observation of the attack is that the secret oil space is an invariant subspace (an eigenspace) of $P'_{ij} = P'^{-1}_j P'_i$, for any two invertible matrices P'_i and P'_j of the map $\mathcal{P}'$, i.e., $P'^{-1}_j P'_i O = O$. The invariant subspace of the matrices can be found efficiently, for example by looking at the characteristic polynomial of one such matrix P'_{ij}. If the characteristic polynomial factors into two irreducible factors C_1 and C_2 of degree v, then the oil space can be found as the kernel of the matrix $C_1(P'_{ij})$ or $C_2(P'_{ij})$. Kipnis and Shamir [KS06] argue that the probability for such a factorization is high, thus obtaining an efficient algorithm for distilling the oil space.

The unbalanced version, with $n > 2m$ [KPG99] was constructed to prevent this attack. Indeed, in this case two matrices P'_i and P'_j do not necessarily map the oil space into the exact same subspace of the vinegar space. Nevertheless, the intersection of $P'_i O$ and $P'_j O$ is still very high. The authors of [KPG99] show the following crucial result.

Theorem 1 ([KPG99]). *Let Q' be an invertible linear combination of the matrices $P'_1, \ldots, P'_m$. Then for any invertible P'_j, the matrix $P'^{-1}_j Q'$ has a non-trivial invariant subspace, which is also a subspace of O with probability at least $\frac{q-1}{q^{2(n-2m)}-1}$.*

When the difference $n-2m$ is not very big, this probability is high, and one can expect to find this subspace efficiently. In order to do that, the original Kipnis-Shamir attack can be generalized to look for a small invariant subspace of the matrices P'_{ij} that is also a subspace of O. Thus, one can use any factorization of the characteristic polynomial of P'_{ij}, and check whether for any of the factors C, the kernel of $C(P'_{ij})$ is in the oil subspace. Recall that we can efficiently test membership of a vector in the oil space by checking whether the public map vanishes on the said vector.

Note that the procedure needs to be repeated, in order to find the whole oil space, or one can use other methods such as the reconciliation attack (see Section 2.3.2), once a few oil vectors are known.

2.3.2 Reconciliation Attack

Recall from Remark 1 that the public map of UOV is (m,m)-linear. Thus, in order to break the scheme, it is necessary to find a vector space - the oil space $\boldsymbol{O}$, such that $\mathcal{P}$ is (m,m)–linear with respect to $(\boldsymbol{O}, \mathbb{F}_q^m)$.

Ding et al. in [DYC+08] propose an algorithm that sequentially performs a change of basis that reveals gradually the space O. They call the algorithm *Reconciliation Attack on UOV*. In Algorithm 5, we present an equivalent version of the attack interpreted in terms of (s,t)–linearity (cf. Algorithm 2 [DYC+08]).

Algorithm 5 Reconciliation Attack on UOV in terms of (s,t)–linearity

Input: UOV public key $\mathcal{P}: \mathbb{F}_q^n \to \mathbb{F}_q^m$.
Output: An oil space $\boldsymbol{O} = \boldsymbol{O}_m$ of dimension m.

1: $\boldsymbol{O}_0 \leftarrow$ the zero-dimensional vector space
2: **for** $k := 1$ to m **do**
3: Find $\mathbf{o}_k = (o_1^{(k)}, ..., o_v^{(k)}, 0, ..., 0, 1_{n-k+1}, 0, ..., 0) \in \mathbb{F}_q^n$, where 1_{n-k+1} denotes that the $(n-k+1)$-th coordinate is 1, by solving

$$\begin{aligned} \mathbf{o}_j P'_i \mathbf{o}_k &= 0, \quad \text{for } i \in \{1,\dots,m\} \text{ and } j < k \\ \mathbf{o}_k P_i \mathbf{o}_k &= 0, \quad \text{for } i \in \{1,\dots,m\}. \end{aligned}$$

4: Construct the space $\boldsymbol{O}_k = \boldsymbol{O}_{k-1} \oplus \mathit{Span}\{\mathbf{o}_k\}$
5: **end for**
6: **return** $\boldsymbol{O} = \boldsymbol{O}_m$

Note that the form of the oil vectors $\mathbf{o}_k$ is chosen to assure they are all independent, i.e., the algorithm finds a basis of the oil space. Other forms are possible, and they are all equivalent up to some column permutation. Actually, there is a small probability that a chosen form can't be the basis of the oil space, in case of which we choose randomly another form, i.e., we perform a randomization of the coordinates.

At the k-th iteration, the algorithm solves a system of m quadratic and $(k-1)m$ linear equations in v variables. This means that in the first iteration, there are no available linear relations, so finding the first oil vector is computationally the dominating step. At each subsequent step, we have m additional linear relations that basically reduce the problem to solving a quadratic system of m less variables. As soon as the algorithm reaches $k > v/m + 1$, there are enough linear equations to solve the system, and finding the rest of the oil vectors becomes easy.

Remark 3. The given description of the reconciliation attack from Algorithm 5 is very similar to the recent description given by Beullens [Beu21].

3 Strategy for a Complete Secret Key Recovery

In this section, we summarize the attack strategy. First, we highlight a correspondence between private and public keys in UOV, invoked by the design choices of the scheme. This correspondence partially reveals the input of the subroutine that is the target of our side-channel attack. We explain what information we get from the power measurements and how to exploit them to recover a vector from the secret oil space. Finally, we add a brief complexity analysis of the reconciliation attack with one (or two) known oil vectors, which is used to obtain the remaining oil space and therefore, the complete secret key.

3.1 Overlap in Public and Private Key

One possible way to generate a valid UOV key pair is given in Algorithm 1. This is what we refer to as 'Standard UOV' in the following, as there are no compression techniques applied and the secret key can be used for signing right away. It is also possible to not store large parts of the coefficients of the matrices and instead either expand them from a seed or calculate back and forth between them via Equation (2). Thereby, the key sizes can be reduced massively, at the expense of signing and verification time. This will be noted as 'Compressed UOV'. We will now show, that due to the relation given by Equation (2), in both cases[1], the entries of the sub-matrices $F_1^{(i)}$ of Equation (7) are obvious to any person with access to the public key.

Standard UOV First, the secret key $sk = (T, F^{(1)}, \ldots, F^{(m)})$ is randomly generated. The matrices are of the block-matrix structure given in Equations (7) and (3). Now, the public key $pk = (P^{(1)}, \ldots, P^{(m)})$ is computed by evaluating $P^{(i)} = T^\top F^{(i)} T$ and bringing the resulting matrices to upper triangular form. Since the blocks $F_1^{(i)}$ are already upper triangular matrices, this operation has no impact on them. From

$$\begin{pmatrix} I & 0 \\ T_1^\top & I \end{pmatrix} \begin{pmatrix} F_1^{(i)} & F_2^{(i)} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} I & T_1 \\ 0 & I \end{pmatrix} = \begin{pmatrix} F_1^{(i)} & F_1^{(i)} T_1 + F_2^{(i)} \\ T_1^\top F_1^{(i)} & T_1^\top F_1^{(i)} T_1 + T_1^\top F_2^{(i)} \end{pmatrix}$$

we deduce

$$P^{(i)} = \begin{pmatrix} P_1^{(i)} & P_2^{(i)} \\ 0 & P_3^{(i)} \end{pmatrix} = \begin{pmatrix} F_1^{(i)} & (F_1^{(i)} + F_1^{\top(i)}) T_1 + F_2^{(i)} \\ 0 & \texttt{Upper}(T_1^\top F_1^{(i)} T_1 + T_1^\top F_2^{(i)}) \end{pmatrix}. \tag{8}$$

We notice that the special structure of T leads to $P_1^{(i)} = F_1^{(i)}$, which implies that a considerable amount of the public and private key is identical.

Compressed UOV Here the order is reversed. First, the matrices $P_1^{(i)}$ and $P_2^{(i)}$ are expanded from a random seed pk_{seed}. Then, after T is randomly generated from a secret seed sk_{seed}, the relation shown in Equation (8) is used to compute $F_1^{(i)}$ and $F_2^{(i)}$. Thus, the secret key only consists of two seeds $sk = (pk_{seed}, sk_{seed})$. Finally $P_3^{(i)}$ is computed, again following Equation (8) to complete the key generation. Note, that again $P_1^{(i)} = F_1^{(i)}$ is satisfied. This time $P_1^{(i)}$ is not directly included in the public key, but can be recovered by expanding the public key seed pk_{seed}, which is part of the public key $pk = (pk_{seed}, P_3^{(i)})$.

[1] We want to point out that there are more possible ways to generate valid UOV key pairs. We will bring up one of them in Section 5, where we discuss countermeasures. We focus on these two in the attack description, as they are mainly considered and employed by current implementations.

3.2 Single-Trace Recovery of the Vinegar Variables

The previous findings help us to identify a vulnerability in terms of side-channel resistance in the signing process. Namely, it is the sub-routine responsible for setting up the constant part of the system of linear equations, indicated in Section 2.2. It is given by computing

$$\tilde{\boldsymbol{x}} F_1^{(k)} \tilde{\boldsymbol{x}}^\top = (\tilde{x}_1, \ldots, \tilde{x}_v) \begin{pmatrix} \alpha_{1,1}^{(k)} & \cdots & \alpha_{1,v}^{(k)} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \alpha_{v,v}^{(k)} \end{pmatrix} (\tilde{x}_1, \ldots, \tilde{x}_v)^\top, \tag{9}$$

using the secret key matrices $F_1^{(k)}$, for all $k \in \{1, \ldots, m\}$. To abbreviate, we will write $\tilde{\boldsymbol{x}} \boldsymbol{F_1} \tilde{\boldsymbol{x}}^\top$ if we want to compute Equation (9) for all $k \in \{1, \ldots, m\}$. Here, the randomly generated vinegar variables $(\tilde{x}_1, \ldots, \tilde{x}_v)$ are multiplied with a considerable amount of known values $\alpha_{i,j}^{(k)}$. In more detail, for every $i \in \{1, \ldots, v\}$ the product

$$\alpha_{i,i}^{(k)} \cdot \tilde{x}_i \tag{10}$$

is computed for all $k \in \{1, \ldots, m\}$. The power consumption of this multiplication depends on the exact value of the respective vinegar variable, which makes them an apparent target for power analysis. In Section 4, we present the details of the suggested side-channel attack. In fact, the attack vector is so strong, that we are able to recover all v vinegar variables from measuring the power consumption of just one signing process with large probability. In the next section, we show that a whole set of vinegar variables, together with the corresponding signature leads to a secret oil vector.

3.3 Obtaining a Secret Oil Vector

As stated in Section 2.2, the vinegar variables $\tilde{\boldsymbol{x}} = (\tilde{x}_1, \ldots, \tilde{x}_v)$ in combination with the secret key generate a linear system of equations. Its solution $\boldsymbol{x'} = (x_{v+1}, \ldots, x_{v+m})$, the so-called oil variables, are concatenated to the vinegar variables. To finalize the signature generation, the resulting vector $(\tilde{\boldsymbol{x}}, \boldsymbol{x'})^\top$ is transformed by T^{-1}, as depicted in Line 9 of Algorithm 2. The result constitutes the signature $(\boldsymbol{s_1}, \boldsymbol{s_2})^\top$. Thus, it holds

$$\begin{pmatrix} \boldsymbol{s_1} \\ \boldsymbol{s_2} \end{pmatrix} = \begin{pmatrix} I & -T_1 \\ 0 & I \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{x}} \\ \boldsymbol{x'} \end{pmatrix} = \begin{pmatrix} \tilde{\boldsymbol{x}} - T_1(\boldsymbol{x'}) \\ \boldsymbol{x'} \end{pmatrix}.$$

Obtaining $\tilde{\boldsymbol{x}}$ by a side-channel attack, enables us to choose a vector $\boldsymbol{o} = (T_1(\boldsymbol{x'}), \boldsymbol{x'})^\top$ with the property, that its first $n - m = v$ entries are zero, after it is transformed by T. From the structure of the secret key matrices F_i in Equation (8) one can see that $\mathcal{P}(\boldsymbol{o}) = \mathcal{F} \circ T(\boldsymbol{o}) = 0$. Consequently, we found a vector of the secret oil space $\boldsymbol{o} \in \boldsymbol{O}$. This reduces the complexity of algebraic key recovery attacks significantly.

Remark 4. The previous section basically concludes that if one is in possession of a signature and the corresponding vinegar vector used to build this signature, one is able to determine a secret oil vector by subtracting the two from each other. Using the description given in Section 2.1.2, this is quite obvious, since the signature is of the form $\boldsymbol{s} = \boldsymbol{v} + \boldsymbol{o}$. However, the implementations currently considered follow the description in Section 2.1.1, so we added the former result for clarification.

3.4 Recovering the Secret Oil Space Using (a Combination of) the Kipnis-Shamir Attack and the Reconciliation Attack

As discussed in Section 2.3.2 (also illustrated in [Beu22b, Section 4.1]), when one oil vector $\mathbf{o}_1$ is known in the reconciliation attack, we obtain m linear equations in the entries of a

second $\mathbf{o}_2$. Thus finding a second vector $\mathbf{o}_2 \in O$ can be achieved by solving a quadratic system of m equations in $n-m-m$ variables which can be done using any general system solver, for example the XL algorithm and its variants [CKPS00, Die04, YC05], the (Hybrid) F4/F5 algorithm [Fau99, Fau02, BFP12] or the Joux-Vitse Crossbred algorithm [JV17]. Since $n = 2,5m$, the complexity of the attack is significantly reduced, however, it is still exponential in the number of variables. Instead of using an algebraic solver and the reconciliation attack, we can first use a modification of the Kipnis-Shamir attack to find a second vector $\mathbf{o}_2 \in \boldsymbol{O}$. Alternatively, we can obtain $\mathbf{o}_2$ by an SCA in the same way we recovered $\mathbf{o}_1 \in \boldsymbol{O}$. Note that a second oil vector again provides us with m linear equations and $n < 3m$, so once two oil vectors are known, the remaining oil space can be recovered in polynomial time using the reconciliation attack. We measured the time consumption of this step with the parameters of different security levels and present it in the last column of Table 1. We describe next the procedure for recovering the second oil vector using the Kipnis-Shamir attack.

Using the known oil vector $\mathbf{o}_1$, we form the linear equations $\mathcal{P}'(\mathbf{o}_1, \mathbf{x}) = 0$, i.e.,

$$\mathbf{o}_1 P_i' \mathbf{x} = 0, \quad \text{for } i \in \{1, \ldots, m\}$$

that characterize the space of the remaining oil vectors. We can use these to replace m variables $x_{n-m+1}, \ldots, x_n$ in the equations of the public system $\mathcal{P}(\mathbf{x}) = 0$. Furthermore, for a nonzero coordinate $i \in \{1, \ldots, n-m\}$ of the oil vector $\mathbf{o}_1$, fixing $x_i = 0$ restricts the oil space of the system to $O \setminus \{\mathbf{o}_1\}$. Without loss of generality, let $i = n-m$. We can now apply efficiently the Kipnis-Shamir attack on the obtained system as long as the number of variables $n-m-1$ is close to two times the dimension of the oil space $dim(O \setminus \{\mathbf{o}_1\}) = m-1$.

However, the attack can't be directly applied, because the symmetric matrices of the obtained system are not full rank. As a matter of fact, the rank is only $2n-4m$. The reason is that for each P_i, the linear equations above add $m-1$ additional constrains on the oil $\times$ vinegar part, which means the kernel of the matrices is at least $2m-1-(n-m) = 3m-n-1$. This situations can be remedied, by fixing additional $3m-n-1$ variables, but this will also reduce the dimension of the oil space by $3m-n-1$. In total, we obtain a system

$$\mathcal{M}_i(x_1, \ldots, x_{2n-4m}) = 0, \ i \in \{1, \ldots, m\}$$

whose oil space $O_{\mathcal{M}}$ is of dimension $n-2m$, i.e., we obtain a balanced oil and vinegar instance, for which the Kipnis-Shamir attack works best.

As said above, we only need to find one additional oil vector, because then the reconciliation attack can very efficiently recover the rest of the oil space O. Thus, the generalization of Kipnis-Shamir attack from [KPG99] for finding a small invariant subspace of $M_j'^{-1} M_i'$ works best, and we expect to find an oil vector in approximately q trials.

We have implemented and verified both parts of the algebraic attack for all NIST security levels. The results are summarized in Table 1.

Table 1: Practical experiments on different parameter sets when only one oil vector is available. We show the time for finding a second oil vector with the KS attack, and the time for finding $m-2$ more basis vectors of O with the reconciliation attack.

Security level	Kipnis-Shamir step in *hh*:*mm*:*ss*	Reconciliation step in *hh*:*mm*:*ss*
Reduced parameters $(v, m) = (42, 28)$	00:00:25	00:01:07
Security level I $(v, m) = (68, 44)$	00:08:00	00:11:34
Security level II $(v, m) = (112, 72)$	00:44:36	02:23:19
Security level III $(v, m) = (148, 96)$	00:58:16	10:42:51

4 Executing the Side-Channel Attack

For the side-channel attack presented here, we exploit the data-dependent power consumption of the subroutine discussed in Section 3.2 to compute $\tilde{\boldsymbol{x}}\boldsymbol{F_1}\tilde{\boldsymbol{x}}^\top$. In [PSKH18, PSN21], the authors have already published side-channel attacks on UOV and Rainbow based on correlation power analysis (CPA). However, in all these attacks, the attacker model assumes some kind of public data under control, in order to perform a classic CPA with different messages, i.e., the attack needs control over the digest of the salted and hashed message $\mathcal{H}(\mathcal{H}(d)||salt)$. Furthermore, their attacks require a considerable amount of power traces, taken from the target device. As described in [PSKH18, Section 4.1] around 30 traces are needed to recover elements of the secret linear transformation. Moreover, they are attacking a generic matrix-vector multiplication deviating from the specific implementation that is used in recent UOV implementations.

Our proposed side-channel attack does not require control over the message d during the attack phase to extract vinegar variables. Instead of conducting a CPA, we perform a template-based profiling attack to extract individual bits of the processed vinegar variables while performing the operation $\tilde{\boldsymbol{x}}\boldsymbol{F_1}\tilde{\boldsymbol{x}}^\top$. Therefore, we need a learning phase with access to an identical device running the same implementation of UOV that we are targeting. We also investigate a transferable SCA scenario where the training device in the template building phase is not identical to the target device in the attack phase. We used two STM32F3 core-based platforms, and were also able to extract the secret $\tilde{\boldsymbol{x}}$ in the attack phase successfully. In the learning phase, we recorded reference traces for every possible value in $\mathbb{F}_q$, i.e., in our case $q = 256$. We fix every entry in $\boldsymbol{c} \in \mathbb{F}_q^v$ to the same element in $\mathbb{F}_q$ and execute the operation $\boldsymbol{c}\boldsymbol{F_1}\boldsymbol{c}^\top$ a single time. Subsequently, for every entry $i \in \{1, \ldots, v\}$, we cut out the region of the trace where the entry $c_i \in \boldsymbol{c}$ is processed, so that it can be compared to the corresponding region of $x_i \in \tilde{\boldsymbol{x}}$ that is measured in the attack phase. Because of the dependencies between the private and public keys discussed in Section 3.1, we know the α-values of the secret key of the signature operation we are targeting by considering the available public key. Hence, we can create a template for the multiplication operation in Equation (10) with all the necessary knowledge.

In detail, we exploit execution patterns in the power consumption caused by a bit-value dependent execution within the bit-sliced implementation of the targeted operation. In Section 4.2, we discuss the side-channel exploitation of the implementation in detail.

4.1 Practical Setup

All practical experiments were implemented using the ChipWhisperer tool chain [Tec23] (version 5.6.1) in Python (version 3.8.10) and performed on a ChipWhisperer-Lite board with the CW308T UFO board. The victim board, containing a 32-bit STM32F303RCT7 microcontroller with ARM Cortex-M4 architecture, is mounted on the UFO board. The ARM Cortex-M4 implementation was compiled with `arm-none-eabi-gcc` (version 10.1.0) and the C reference implementation for x86 compiled with `gcc` (version 10.1.0). For running the experiments on our setup, slight modifications of the available implementations [UOV23] were required. The side-channel exploited sections of the code remain unchanged.

Due to the SRAM size of 40 KB on the target STM32F3 core, we specify a reduced parameter set aimed at adapting the attacked routine to the limitations of the target device. For this, we chose the parameters $(v, m) = (42, 28)$ as shown in Table 2 resulting in ≈ 25 KB of used memory for the matrices in $\boldsymbol{F_1}$. Further, our ARM implementation only includes the required functions from Section 3.2 for generating traces. Thus, the execution time for a single run is reduced and more experiments can be conducted within a certain time frame. We note that by adapting the parameter set, our implementation can be used to attack other instantiations, e.g., higher security levels (see Table 2) on suitable target boards. Every coefficient is a field element in $\mathbb{F}_q$ and thus, consumes one byte if $q = 256$.

Table 2: Required memory size for computing $\tilde{\boldsymbol{x}}\boldsymbol{F_1}\tilde{\boldsymbol{x}}^\top$ for different security levels.

Security level	Number of coefficients in $\boldsymbol{F_1}$ $m \cdot v \cdot (v+1)/2$
Reduced parameters $(v, m) = (42, 28)$	25.284
Security level I $(v, m) = (68, 44)$	103.224
Security level II $(v, m) = (112, 72)$	455.616
Security level III $(v, m) = (148, 96)$	1.058.496

Additionally, we adapted the C reference implementation for the x86 architecture from [UOV23] in order to output the public key and signature in a file, since we need them for the attack later on. Furthermore, we output the part of the public key, that contains the coefficients of $\boldsymbol{F_1}$, and the used vinegar variables $\tilde{\boldsymbol{x}}$ in another file. Note, that the latter are used solely to execute the function from Equation (9) on the target device. They are not used in the analysis of our measurements, nor in the subsequent algebraic attack, since they are not known to an attacker. The reference traces are recorded on a profiling device identical to the target device. Hereby, the used vinegar variables are fixed to a certain value $c \in \mathbb{F}_q$ while recording the corresponding trace.

4.2 Exploitable Side-Channel Information

The vinegar variables $\tilde{\boldsymbol{x}}$ are passed via the function parameter `uint8_t b` to the function `gf256v_mul_u32(uint32_t a, uint8_t b)` and get internally processed bitwise. This function performs the basic multiplication operation stated in Equation (10). This operation is required during the computation of $\tilde{\boldsymbol{x}}\boldsymbol{F_1}\tilde{\boldsymbol{x}}^\top$ to set up the constant part of the generated linear system during signing. Then, depending on the value of each of the 8 bits of $\tilde{x}_i$ (resp. parameter `uint8_t b`), a multiplication will be executed, see Line 2 in Algorithm 6 or, for more details, Line 5, 10, 15, 20, 25, 30, 35, 40 in Listing 1 in the appendix. As shown in Figure 1, the spots for these multiplications are easily recognizable for every single bit within the captured power traces.

Algorithm 6 Algorithmic representation of `gf256v_mul_u32(`$\alpha^k_{i,i}, \tilde{x}_i$`)` from [UOV23]

Input: $\alpha^k_{i,i}, \tilde{x}_i$
Output: $\alpha^k_{i,i} \cdot \tilde{x}_i$
1: **for** $z_0 = 0$ `to` 7 **do**
2: temp = temp **xor** $\alpha^k_{i,i} \cdot ((\ \tilde{x}_i \ggg z_0)$ **and** 1)
3: α_msb = $\alpha^k_{i,i}$ **and** 0x80808080
4: $\alpha^k_{i,i} = \alpha^k_{i,i}$ **xor** α_msb
5: $\alpha^k_{i,i} = (\alpha^k_{i,i} \lll 1\ )$ **xor** $((\ \alpha^k_{i,i}$_msb $\ggg 7\ ) \cdot \tilde{x}_i)$
6: **end for**
7: **return** temp

In the learning phase of our attack, we collect and store reference power traces for all 256 possible values of `uint8_t b`, i.e., for the values $\tilde{x}_i$. The collected traces represent the average over $m/4$ runs (since four α-values are processed simultaneously) and include 1000 analog digital converter (ADC) samples recorded in a single capture during the execution of the function `gf256v_mul_u32`. As shown in Figure 1, this number of ADC samples is sufficient to capture the entire execution of the targeted function. The reference traces must be captured only once for a certain target device and public key, and can be reused for further attacks.

After collecting the reference traces, we can create template traces based on the reference traces by extracting and concatenating the points of interest. Thus, the template

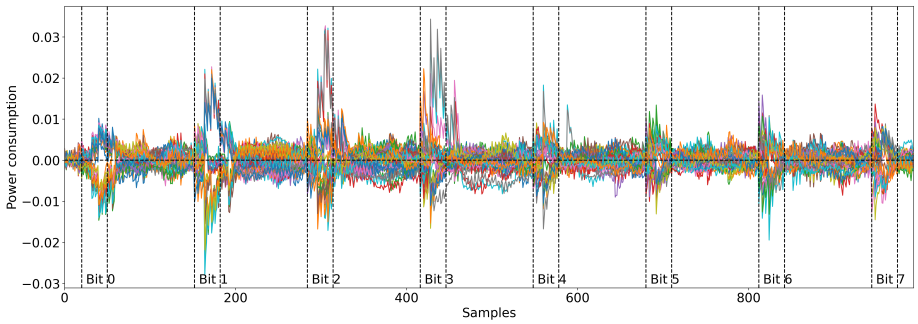

Figure 1: Captured power traces during the execution of the attacked function `gf256v_mul_u32(uint32_t a, uint8_t b)` for random `b` for all 42 vinegar variables.

trace consists only of the samples marked in Figure 1. As can be seen from Figure 1, we obtain for each of the possible 256 vinegar candidates one unique sequence of value per reference trace based on each processed bits. With this construction of a template, we can perform a correlation analysis that follows more the idea of a horizontal attack as known from CPA attacks against elliptic-curve cryptography (ECC), cf. [NC17].

In the leak extraction phase, usually referred to as the attack phase in the context of a side-channel template attack, we use Pearson correlation [BCO04] to compare all generated reference traces with the acquisition power consumption of a signature creation operation. Of course, we also need to prepare the recorded power traces similarly to the reference traces and extract only the time points of interest of the captured trace for each processing of vinegar variable within the operation of $(\alpha_{i,i}^{(k)} \cdot \tilde{x}_i)$. As shown in Figure 2, the correct candidate can be extracted by calculating the Pearson correlation coefficient between the reference trace and the captured power trace section where the vinegar variable under investigation is processed.

As in general for single trace attacks the extractable information are quite limited. Thus, the correct candidate might not be clearly identifiable due to the signal-to-noise ratio, since there might be one or more field elements that have corresponding power traces with correlation coefficient similar high to the maximum. There is of course no way to verify the correctness of the vinegar variables guess individually, but as described in Section 3.3 there is an efficient way to check if all vinegar variables are correct. We subtract them from the signature and test if the corresponding oil vector candidate is annihilated by $\mathcal{P}$. If this is not the case, we apply a small trial-and-error replacement, where we successively substitute vinegar variables that have a few reference traces with similar high correlation. Following this strategy, we managed to find a valid oil vector, even if our initial guess that uses the values with maximum correlation coefficients was not correct. We implemented the trial-and-error replacement such that it aborts after a few seconds, since we wanted to keep the analysis fast and it was enough in most of the cases. If we have not found an oil vector by then, the attack is considered as unsuccessful, which happened in around 2% of the cases.

4.3 Generalizing the Attack

The achieved results depend strongly on the analyzed implementation, which is shown in Listing 1. The given algorithm processes the targeted vinegar variables bitwise, which implies that the power consumption is directly related to the single bits of the value we want to recover. In the following, we assume that there is a protected implementation in place, that disallows us to draw conclusions about the single bits or even the Hamming weight (HW) of the vinegar variables $\tilde{x}_i$ itself. We will now discuss theoretically a more

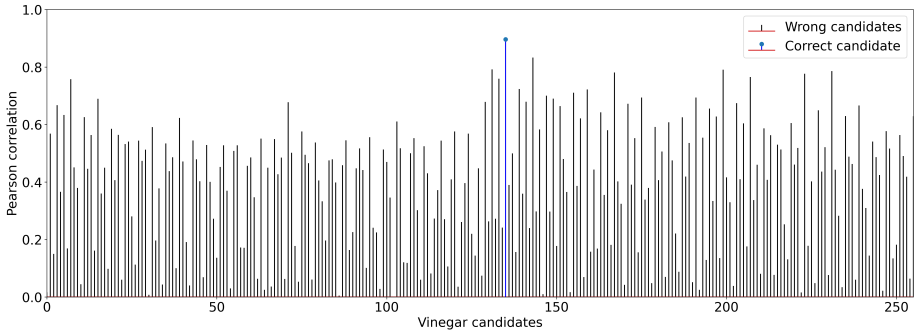

Figure 2: Recovering the value of one vinegar variable by correlation value.

relaxed attack model in which we can only obtain information about the Hamming weight of the products $\alpha_{i,i}^{(k)} \cdot \tilde{x}_i$ via side channels. We prove that this would still allow us to recover the vinegar variables and perform the remaining algebraic attack as described in Section 3, up to a certain noise level.

Focus on a specific entry $\tilde{x}_i$, assuming we have m measurements $w^{(k)} = \mathrm{HW}(\alpha_{i,i}^{k} \cdot \tilde{x}_i)$ and knowledge of the respective values $\alpha_{i,i}^{k}$ for $k = 1, \ldots, m$. Then, this entry can be recovered very precisely, since for every k, there is only a small number of field elements $c \in \mathbb{F}_q$, such that $\alpha_{i,i}^{k} \cdot c$ has the measured Hamming weight $w^{(k)}$. A natural method to recover the right value of $\tilde{x}_i$, is to go through $k = 1, \ldots, m$ and count for every $c \in \mathbb{F}_q$ the number of times the Hamming weight of the product $\alpha_{i,i}^{k} \cdot c$ does not coincide with the measured Hamming weight. The element with the smallest number of misses is likely to be the $\tilde{x}_i$ we are looking for.

We have carried out several simulations to test at which noise levels the described method has a high probability of success. To simulate a certain noise level, we took the correct Hamming weights and added an error value $\epsilon^{(k)}$ following a normal distribution with deviation σ. Consequently, adding error values following such a distribution with, e.g., $\sigma = 1$, implies around 68.3% of the conducted Hamming weight measurements $w^{(k)}$ are correct and the others are faulty. Table 3 summarizes the results for various noise levels and the parameter sets of three different security levels[2]. Here, p_i states the probability for correctly obtaining one vinegar variable $\tilde{x}_i$ and $\boldsymbol{p}$ states the probability for a successful recovery of the complete vinegar vector $\tilde{\boldsymbol{x}}$.

Table 3: Success probability of recovering the right vinegar variable(s) at different noise levels of the generalized Hamming weight attack.

Noise level		Security level I $(v, m) = (68, 44)$		Security level II $(v, m) = (112, 72)$		Security level III $(v, m) = (148, 96)$	
dev	acc in %	p_i	$\boldsymbol{p}$	p_i	$\boldsymbol{p}$	p_i	$\boldsymbol{p}$
$\sigma = 1$	68.3	99.9	$93,3$	> 99.9	> 99.9	> 99.9	> 99.9
$\sigma = 1.3$	55.8	98.8	43.3	99.9	93.3	> 99.9	> 99.9
$\sigma = 1.4$	52.5	97.9	20.0	99.8	83.3	99.9	93.3

There are two main conclusions we can draw from this. First, even if only around two thirds of the conducted Hamming weight measurements $w^{(k)}$ are correct, we can still recover a whole vinegar vector with high probability. This can be transformed to an oil vector, that enables an efficient Kipnis-Shamir attack, similar to our strategy in Section 3. Second, this probability even grows for larger parameter sets, since there are more available

[2]The script for simulating the noise and determining the success probability of the recovery of the used values can also be found at https://github.com/mstoetti/SCA_assisted_recon_UOV.

measurements for each vinegar variable, as m grows.

These are just theoretical bounds stating what noise level would be acceptable to still run the generalized attack on the Hamming weight of the products $\alpha^k_{i,i} \cdot \tilde{x}_i$. The real noise level would depend on the target architecture and on the concrete implementation that is chosen to protect the vinegar variables from our attack in Section 4.2.

5 Countermeasures

In this section, we discuss possible countermeasures to prevent the side-channel attack proposed in this work. We will discuss in total four countermeasures, one of which is specific to the UOV algorithm while the other countermeasures rely on established techniques. The concrete implementation of the countermeasures is not discussed in this section and is reserved for future work.

5.1 UOV-Specific Countermeasure

The attack described in this work relies on the correspondence between the public key and the private key, which results from the choice of the linear transformation T (Section 3.1, Equation (3)). This kind of key generation has been coined as *equivalent keys* and is the usual way of generating UOV keys. Another method exists, however, which is referred to as *random affine T* [PSKH18]. In this method the whole matrix T is generated randomly such that it does not feature three blocks of all zeros or the identity matrix. When T is chosen like this, both the $(\alpha^k_{i,j})$ and the vinegar variables are secret. Hence, the attacker can observe side-channel leakage of only a multiplication of two unknown factors. In the following, we show how our attack adapts to this case. We show that, since both the vinegar variables $\tilde{x}_i$ and the secret key elements $\alpha^k_{i,j}$ have to be revealed, an attacker needs considerably more side-channel traces and the attack is less noise-resistant. Still, an attacker can reveal secret information.

In our target routine, the first operation performed is $\boldsymbol{F_1} \cdot \tilde{\boldsymbol{x}}^\top$, followed by $\tilde{\boldsymbol{x}} \cdot \boldsymbol{F_1} \cdot \tilde{\boldsymbol{x}}^\top$. The intermediate results of the first matrix vector multiplication are accumulated in the considered implementation:

$$F_1^{(k)} \cdot \tilde{\boldsymbol{x}}^\top = \begin{pmatrix} \alpha^k_{1,1}\tilde{x}_1 & + & \alpha^k_{1,2}\tilde{x}_2 & + & \cdots & + & \alpha^k_{1,v}\tilde{x}_v \\ & & \alpha^k_{2,2}\tilde{x}_2 & + & \cdots & + & \alpha^k_{2,v}\tilde{x}_v \\ & & & \ddots & & + & \vdots \\ & & & & & & \alpha^k_{v,v}\tilde{x}_v \end{pmatrix}.$$

In a second step, $\tilde{\boldsymbol{x}}$ is multiplied to this vector from the left:

$$\tilde{\boldsymbol{x}} \cdot F_1^{(k)} \cdot \tilde{\boldsymbol{x}}^\top = \left(\sum_{j=1}^{v} \alpha^k_{1,j}\tilde{x}_j \right) \tilde{x}_1 + \left(\sum_{j=2}^{v} \alpha^k_{2,j}\tilde{x}_j \right) \tilde{x}_2 + \cdots + \alpha^k_{v,v}\tilde{x}_v\tilde{x}_v.$$

During both computations, we can observe the Hamming weight of all individual summands, i.e., the product of two or three unknown values. This is specific to the implementation we are attacking in this work, which is the latest implementation of UOV [UOV23]. Other implementations might lead to other exploitable information.

The general idea underlying this attack is that the Hamming weight of a product carries information about potential values of the factors. First, we focus on the HW pairs $HW(\tilde{x}_v \cdot \alpha^k_{v,v})$ and $HW(\tilde{x}_v \cdot \tilde{x}_v \cdot \alpha^k_{v,v})$,

corresponding to the last vinegar variable $\tilde{x}_v$. Hence, by analyzing these Hamming weights, we learn the HW of a product $(\tilde{x}_v \cdot \tilde{x}_v \cdot \alpha^k_{v,v})$ and also of one of the factors $(\tilde{x}_v \cdot \alpha^k_{v,v})$, which helps us reveal information about the co-factor $\tilde{x}_v$. We emphasize that

both multiplications are computed for m different values $\alpha_{v,v}^k$, i.e., all multiplications using the same vinegar vector $\tilde{\boldsymbol{x}}$ are computed with m different secret matrices. This increases the information one gets about the vinegar variables. By passing through all provided $k = 1, \ldots, m$ pairs, we can step by step reduce the number of candidates and eventually reduce the number of candidates to a small number (between 1 and 3) with high probability. Consequently, we can almost uniquely learn the value of the vinegar variable $\tilde{x}_v$.

In the next step, we repeat this analysis for several signatures, which is why more side-channel traces are needed in this scenario. Each new signature uses different vinegar variables, but the same matrices $(\alpha_{i,j}^k)$. Hence, we obtain different sets of candidates for the last vinegar variables $\tilde{x}_v^{(i)}$ used to generate the corresponding signature. Together with the knowledge of the weights

$$HW(\tilde{x}_v^{(i)} \cdot \alpha_{j,v}^1), \ldots, HW(\tilde{x}_v^{(i)} \cdot \alpha_{j,v}^m)$$

for all i and j, these candidates facilitate the recovery of all $\alpha_{j,v}^1, \alpha_{j,v}^2, \ldots, \alpha_{j,v}^m$, for every j, i.e., the last column of all the matrices $(\alpha_{i,j}^k)$ with $k \in \{1, \ldots, m\}$. Assuming that we choose the amount of signatures high enough, which also highly depends on the noisiness of the traces, this will leave us with unique values $\alpha_{j,v}^l$ instead of only a list of candidates. With these exact values we can revisit the list of candidates for $\tilde{x}_v^{(i)}$ and also obtain the exact value for $\tilde{x}_v^{(i)}$ that meets the requirement for the corresponding Hamming weights. We proceed to the previous column, and so on. In general for column $v - s$:

1. Recover first the value of $\tilde{x}_{v-s}^{(i)}$ (almost uniquely) from the knowledge of the Hamming weight of the first summand, the exact knowledge of the remaining summands, and the Hamming weight of the product of $v_{v-s}^{(i)}$ with this sum:

$$\begin{array}{ll} HW(\tilde{x}_{v-s}^{(i)} \cdot \alpha_{v-s,v-s}^1) + \cdots + \tilde{x}_v^{(i)} \cdot \alpha_{v-s,v}^1 & HW(\tilde{x}_{v-s}^{(i)}(\tilde{x}_{v-s}^{(i)} \cdot \alpha_{v-s,v-s}^1 + \cdots + \tilde{x}_v^{(i)} \cdot \alpha_{v-s,v}^1)) \\ HW(\tilde{x}_{v-s}^{(i)} \cdot \alpha_{v-s,v-s}^2) + \cdots + \tilde{x}_v^{(i)} \cdot \alpha_{v-s,v}^2 & HW(\tilde{x}_{v-s}^{(i)}(\tilde{x}_{v-s}^{(i)} \cdot \alpha_{v-s,v-s}^2 + \cdots + \tilde{x}_v^{(i)} \cdot \alpha_{v-s,v}^2)) \\ \vdots & \vdots \\ HW(\tilde{x}_{v-s}^{(i)} \cdot \alpha_{v-s,v-s}^m) + \cdots + \tilde{x}_v^{(i)} \cdot a_{v-s,v}^m & HW(\tilde{x}_{v-s}^{(i)}(\tilde{x}_{v-s}^{(i)} \cdot \alpha_{v-s,v-s}^m + \cdots + \tilde{x}_v^{(i)} \cdot \alpha_{v-s,v}^m)). \end{array}$$

2. As previously, now recover all $\alpha_{j,v-s}^1, \alpha_{j,v-s}^2, \ldots, \alpha_{j,v-s}^m$, for all j.
3. Now, use the $\alpha_{j,v-s}^l$ to determine $\tilde{x}_{v-s}^{(i)}$ uniquely, for all i.

The previous procedure recovers all vinegar variables and the entire matrices $\boldsymbol{F_1}$. It demands a high number of traces and is very susceptible to noise, but shows that choosing T as affine random T does not completely prevent the attack presented in this work.

5.2 Generic Countermeasures

In contrast to the previous discussed countermeasure that exploits the mathematical structure and properties of UOV the here discussed countermeasure concepts can be applied to various cryptographic schemes. In detail, we will discuss three different possible concepts to prevent our proposed attack. These countermeasure techniques are masking, shuffling and pre-computation. Also we will only focus on prevention of first order side-channel leakage.

Masking is a well-established countermeasure against side-channel attacks and can also be used to prevent our proposed attack. In [PSKH18] the authors already propose a multiplicative masking scheme for the operation of a matrix-vector multiplication. This proposal can be directly applied to the computation of $\alpha_{i,i}^k \cdot \tilde{x}_i$ to prevent first-order leakage exploitation. The basic idea is to multiply a random non-zero value Ψ to the secret vinegar variable $\hat{x}_i = \tilde{x}_i \cdot \Psi$. Hence, the operation $\hat{y}_i = \alpha_{i,i}^k \cdot \hat{x}_i$ is performed with the randomized vinegar variable $\hat{x}_i$. Afterwards, the original value can be reconstructed by

multiplying the inverse of the random value Ψ^{-1} by the multiplication result $y_i = \hat{y}_i \cdot \Psi^{-1}$. In this multiplicative masking setting, an attacker would only obtain the masked vinegar variable $\hat{x}_i$ by applying our template-based CPA to the captured traces. Of course, an attacker could perform our proposed attack twice to first guess the Ψ and then reconstruct $\tilde{x}_i = \hat{x}_i \cdot \Psi^{-1}$. To perform such a second-order template attack, the noise in the power traces must be very low to correctly guess Ψ.
In praxis shuffling is implemented in parallel to masking to decrease the signal-to-noise ration of the captured traces. This noise introduced by shuffling will not cancel the side-channel leakage, but more measurements needs to be conducted to extract the side-channel information. Due to [HOM06] the number of required traces for a successful attack increases quadratically. In case of the presented attack on UOV we can only captured one trace with the same settings of α. Therefore, shuffling can be considered as a sufficient countermeasure against our proposed single trace template attack. To counteract our analysis, shuffling can be applied at two different places. First, shuffling can be applied on the vector-matrix multiplication given in Equation (9). The authors of [PSKH18] already proposed a shuffling scheme on matrix-vector multiplication to prevent their proposed CPA attack. In the attack we introduce, the shuffling approach only needs to be applied to the multiplication involving the diagonal elements of the central map ($\alpha^k_{i,i} \cdot \tilde{x}_i$). Thus, depending on the security parameters of UOV, the number of possible execution sequences corresponds to the factorial of m. Thereby the proposed randomization of the index i does not provide as much permutation as the scheme proposed by [PSKH18], but it requires less randomness to execute the permutation.
Second, execution order of the bit-dependent operation in the bit-sliced implementation of `gf256v_mul_u32` can be shuffled. The proposed shuffle scheme in Algorithm 7 exploits a modulus operation and a random value Φ to change the starting index of the cyclic execution over the 8 bits. Thereby not all potential permutations of the index sequence by !8 are exploited, but it provides a minimal computational overhead. By just applying the alternating starting point of the sequence in Line 2 of Algorithm 7 we have only 8 different sequences.

Algorithm 7 Shuffled conditional move version of Algorithm `gf256v_mul_u32`($\alpha^k_{i,i}, \tilde{x}_i, \Phi$)

Input: $\alpha^k_{i,i}, \tilde{x}_i, \Phi$
Output: $\alpha^k_{i,i} \cdot \tilde{x}_i$

1: **for** $z_0 = 0$ `to` 7 **do**
2: $z_1 = (\ (z_0 + \Phi)$ **mod** $7)$
3: **if** $((\ \tilde{x}_i$ » $z_1)$ **and** $1)$ **then**
4: tempT = tempT **xor** $\alpha^k_{i,i}$
5: tempF = tempF **xor** 0x00
6: **else**
7: tempF = tempF **xor** $\alpha^k_{i,i}$
8: tempT = tempT **xor** 0x00
9: **end if**
10: $\alpha^k_{i,i}$_msb = $\alpha^k_{i,i}$ **and** 0x80808080
11: $\alpha^k_{i,i}$ ^= $\alpha^k_{i,i}$ **xor** $\alpha^k_{i,i}$_msb
12: $\alpha^k_{i,i} = (\alpha^k_{i,i}$ « $1\ )$ **xor** $((\ \alpha^k_{i,i}$_msb » $7\ ) \cdot \tilde{x}_i)$
13: **end for**
14: **return** tempT

Hence, an attacker can reconstruct all 8 possible values form the guessed bits. However, with 8 possible candidates per vinegar variable, the complexity increases to a computational complexity of the 8th power to the numbers of entries in $\tilde{\boldsymbol{x}}$, i.e v.

In addition, we adopted the idea of *Always-Double-and-Add*, known as ECC side-channel countermeasure, to hide the conditional execution of the operation in Line 2 of Algorithm 6. In Algorithm 7 the previous conditional operation of temp = temp **xor** $\alpha_{i,i}^k$ in Line 2 of Algorithm 6 is always executed, but conditionally stored in different registers, see Line 3 to 9. Thereby, conditional execution based leakage does not exist anymore. Still this scheme is attackable with an Adress-PDA [IIT02] to distinguish the storage location of the intermediate values, but therefore more than one measurement is required. In general, we propose to apply a combination of the above mentioned countermeasures to prevent our proposed attack on UOV.

Finally, we observe that if the vinegar variables are generated message independent, then their insertion into the central polynomials, i.e., the vulnerable subroutine we identified, might be part of a precomputation step. In case there is an offline phase in place, where message independent operations are precomputed, like suggested by Shim et al. in [SLK22], then this protects also against our proposed attack, since there is no way anymore to obtain necessary side-channel information.

6 Conclusion

In this paper we present a novel side-channel attack against UOV. It exploits leakage, that appears by inserting the vinegar variables into the secret polynomials. This leakage becomes substantial, since we can inherently deduce a large amount of the coefficients of these polynomial. This facilitates a template attack, where we only need a single attack trace to recover a complete set of vinegar variables (resp. an oil vector). We have implemented the attack with the ChipWhisperer Setup and a STM32F3 target board. The success probability thereof, lies above 97% even though we took the reference traces for the template and the attack traces on different devices, i.e., we separated carefully into profiling and target device. From an attacker point of view, it is easy to verify if the side-channel attack was successful, since the retrieved oil vector is annihilated by the public-key map. With the knowledge of one (single-trace) additional oil vector, we can recover the secret key in polynomial time by means of the Kipnis-Shamir attack and the reconciliation attack.

We theoretically extended our approach to a potentially protected implementation that screens the vinegar variables, where we might have only access to the Hamming weights of the products of the vinegar variables and the coefficients of the secret polynomials. We showed that the attack is still feasible and even has a certain noise resistance. Contrary to existing side-channel attacks on UOV, we do not attack the linear transformation. This indicates that our attack is still viable when the implementation is adapted to the description we stated in Section 2.1.1, which omits the usage of the central map and linear transformation. Here, the vinegar variables are inserted directly into the public key map, so we do not need the correspondence derived in Section 3.1.

This also provides ideas for future work, where we want to apply the presented attack to the MAYO signature scheme [Beu22b]. During signing in MAYO, several sets of vinegar variables are inserted into the public key map, that is very similar to the one used in UOV. This indicates the possibility to recover oil vectors in a similar fashion, and with the parameters used in the available public implementation[3], one known oil vector is enough to start a very efficient reconciliation attack. Furthermore, there might be room for improvement regarding our classification technique. We applied a straight forward template attack, where we just clued together the regions of interest and sought for the trace of the field element that had the highest correlation. We managed to achieve good results on the available implementation, but it might be necessary to apply tools like a machine learning classifier to attack more protected implementations.

[3] https://github.com/WardBeullens/MAYO

Acknowledgement

We want to thank Ward Beullens sincerely for pointing us to the applicability of the Kipnis-Shamir attack in our scenario. In a previous version of this work, we needed two traces to be able to construct a linear system of equations, while a single-trace attack required a more complex analysis phase.
This research work has been funded by the German Ministry of Education, Research and Technology in the context of the project Aquorypt (grant number 16KIS1022).

A Source code of `gf256v_mul_u32`

```
static inline uint32_t gf256v_mul_u32(uint32_t a, uint8_t b) {
    uint32_t a_msb;
    uint32_t a32 = a;
    uint32_t b32 = b;
    uint32_t r32 = a32*(b32&1);  // Exploit Bit 0

    a_msb = a32&0x80808080;
    a32 ^= a_msb;
    a32 = (a32<<1)^((a_msb>>7)*0x1b);
    r32 ^= (a32)*((b32>>1)&1);  // Exploit Bit 1

    a_msb = a32&0x80808080;
    a32 ^= a_msb;
    a32 = (a32<<1)^((a_msb>>7)*0x1b);
    r32 ^= (a32)*((b32>>2)&1);  // Exploit Bit 2

    a_msb = a32&0x80808080;
    a32 ^= a_msb;
    a32 = (a32<<1)^((a_msb>>7)*0x1b);
    r32 ^= (a32)*((b32>>3)&1);  // Exploit Bit 3

    a_msb = a32&0x80808080;
    a32 ^= a_msb;
    a32 = (a32<<1)^((a_msb>>7)*0x1b);
    r32 ^= (a32)*((b32>>4)&1);  // Exploit Bit 4

    a_msb = a32&0x80808080;
    a32 ^= a_msb;
    a32 = (a32<<1)^((a_msb>>7)*0x1b);
    r32 ^= (a32)*((b32>>5)&1);  // Exploit Bit 5

    a_msb = a32&0x80808080;
    a32 ^= a_msb;
    a32 = (a32<<1)^((a_msb>>7)*0x1b);
    r32 ^= (a32)*((b32>>6)&1);  // Exploit Bit 6

    a_msb = a32&0x80808080;
    a32 ^= a_msb;
    a32 = (a32<<1)^((a_msb>>7)*0x1b);
    r32 ^= (a32)*((b32>>7)&1);  // Exploit Bit 7

    return r32;
}
```

Listing 1: Vulnerable bit-sliced multiplication operation used for the calculation of $(\alpha_{i,i}^k \cdot \tilde{x}_i)$ in Equation (1) from the available implementation [UOV23].

References

[AKKM20] Thomas Aulbach, Tobias Kovats, Juliane Krämer, and Soundes Marzougui. Recovering Rainbow's Secret Key with a First-Order Fault Attack. In *Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings*, volume 12174 of *Lecture Notes in Computer Science*. Springer, 2020.

[BC14] Christina Boura and Anne Canteaut. A new criterion for avoiding the propagation of linear relations through an Sbox. In *FSE 2013 - Fast Software Encryption*, LNCS, Singapore, 2014. Springer.

[BCH+23a] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. Oil and Vinegar: Modern Parameters and Implementations. Cryptology ePrint Archive, Report 2023/059, 2023. https://ia.cr/2023/059, accepted for publication at TCHES'23.

[BCH+23b] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. Oil and Vinegar: Modern Parameters and Implementations. Cryptology ePrint Archive, Report 2023/059, 2023. https://ia.cr/2023/059.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[Beu21] Ward Beullens. Improved cryptanalysis of UOV and Rainbow. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 348–373. Springer, 2021.

[Beu22a] Ward Beullens. Breaking Rainbow Takes a Weekend on a Laptop. In *CRYPTO*, volume 13508 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2022.

[Beu22b] Ward Beullens. Mayo: Practical post-quantum signatures from oil-and-vinegar maps. In *International Conference on Selected Areas in Cryptography*, pages 355–376. Springer, 2022.

[BFP12] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Solving polynomial systems over finite fields: improved analysis of the hybrid approach. In Joris van der Hoeven and Mark van Hoeij, editors, *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation – ISSAC '12*, pages 67–74. ACM, 2012. https://hal.inria.fr/hal-00776070/document.

[BFSS13] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic Boolean systems. *Journal of Complexity*, 29(1):53–75, 2013. www-polsys.lip6.fr/~jcf/Papers/BFSS12.pdf.

[BHK+19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ Signature Framework. In *CCS*, pages 2129–2146. ACM, 2019.

[BPSV19] Ward Beullens, Bart Preneel, Alan Szepieniec, and Frederik Vercauteren. LUOV. Technical report, National Institute of Standards and Technology, 2019. available at https://github.com/WardBeullens/LUOV/blob/master/Supporting_Documentation/luov.pdf.

[CFM+20] A. Casanova, J.-C. Faugère, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem. GeMSS. Technical report, National Institute of Standards and Technology, 2020.

[CKPS00] Nicolas Courtois, Er Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807, pages 392–407, 2000. www.iacr.org/archive/eurocrypt2000/1807/18070398-new.pdf.

[DCP+20] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang, Matthias Kannwischer, and Jacques Patarin. Rainbow. Technical report, National Institute of Standards and Technology, 2020.

[Die04] Claus Diem. The XL-algorithm and a conjecture from commutative algebra. In *Advances in Cryptology – ASIACRYPT 2004*, volume 3329, pages 323–337, 2004. https://www.iacr.org/archive/asiacrypt2004/33290320/33290320.pdf.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

[DYC+08] Jintai Ding, Bo-Yin Yang, Chia-Hsin Owen Chen, Ming-Shing Chen, and Chen-Mou Cheng. New Differential-Algebraic Attacks and Reparametrization of Rainbow. In *ACNS*, volume 5037 of *LNCS*, pages 242–257, 2008.

[Fau99] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139:61–88, 1999. http://www-polsys.lip6.fr/~jcf/Papers/F99a.pdf.

[Fau02] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation – ISSAC '02*, pages 75–83. ACM, 2002. http://www-polsys.lip6.fr/~jcf/Papers/F02a.pdf.

[FKNT22] Hiroki Furue, Yutaro Kiyomura, Tatsuya Nagasawa, and Tsuyoshi Takagi. A New Fault Attack on UOV Multivariate Signature Scheme. In *PQCrypto*, volume 13512 of *Lecture Notes in Computer Science*, pages 124–143. Springer, 2022.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *Applied Cryptography and Network Security, 4th International Conference, ACNS 2006, Singapore, June 6-9, 2006, Proceedings*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, 2006.

[HTS11] Yasufumi Hashimoto, Tsuyoshi Takagi, and Kouichi Sakurai. General fault attacks on multivariate public key cryptosystems. In *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, pages 1–18, 2011.

[IIT02] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. Address-bit differential power analysis of cryptographic schemes OK-ECDH and OK-ECDSA. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 129–143. Springer, 2002.

[JV17] Antoine Joux and Vanessa Vitse. A Crossbred Algorithm for Solving Boolean Polynomial Systems. In Jerzy Kaczorowski, Josef Pieprzyk, and Jacek Pomykala, editors, *Number-Theoretic Methods in Cryptology - First International Conference, NuTMiC 2017, Warsaw, Poland, September 11-13, 2017, Revised Selected Papers*, volume 10737 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2017.

[KL19] Juliane Krämer and Mirjam Loiero. Fault Attacks on UOV and Rainbow. In *COSADE*, volume 11421 of *Lecture Notes in Computer Science*, pages 193–214. Springer, 2019.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar Signature Schemes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 1999.

[KS06] Aviad Kipnis and Adi Shamir. Cryptanalysis of the oil and vinegar signature scheme. In *Advances in Cryptology—CRYPTO'98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27, 1998 Proceedings*, pages 257–266. Springer, 2006.

[MIS20] Koksal Mus, Saad Islam, and Berk Sunar. QuantumHammer: a Practical Hybrid Attack on the LUOV Signature Scheme. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1071–1084, 2020.

[NC17] Erick Nascimento and Lukasz Chmielewski. Horizontal Clustering Side-Channel Attacks on Embedded ECC Implementations (Extended Version). *IACR Cryptol. ePrint Arch.*, page 1204, 2017.

[PFH+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022.

[PSKH18] 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 Transactions on Cryptographic Hardware and Embedded Systems*, pages 500–523, 2018.

[PSN21] David Pokorný, Petr Socha, and Martin Novotný. Side-Channel Attack on Rainbow Post-Quantum Signature. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 565–568. IEEE, 2021.

[SG14] Simona Samardjiska and Danilo Gligoroski. Linearity Measures for Multivariate Public Key Cryptography. In *SECURWARE 2014, The Eighth International Conference on Emerging Security Information, Systems and Technologies*, pages 157–166, 2014.

[SLK22] Kyung-Ah Shim, Sangyub Lee, and Namhun Koo. Efficient Implementations of Rainbow and UOV using AVX2. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 245–269, 2022.

[Tec23] NewAE Technology. Repository of chipwhisperer tool chain - commit 6bf3bac, 2023. https://github.com/newaetech/chipwhisperer.

[TPD21] Chengdong Tao, Albrecht Petzoldt, and Jintai Ding. Efficient Key Recovery for All HFE Signature Variants. In *CRYPTO (1)*, volume 12825 of *Lecture Notes in Computer Science*, pages 70–93. Springer, 2021.

[UOV23] Repository of Oil and Vinegar: Modern Parameters and Implementations - commit eeedc68, 2023. https://github.com/pqov/pqov-paper.

[VP20] Ricardo Villanueva-Polanco. Cold Boot Attacks on LUOV. *Applied Sciences*, 10:4106, 06 2020.

[YC05] Bo-Yin Yang and Jiun-Ming Chen. All in the XL family: Theory and practice. In Choon sik Park and Seongtaek Chee, editors, *Information Security and Cryptology – ICISC 2004*, pages 67–86, 2005. http://by.iis.sinica.edu.tw/by-publ/recent/xxl.pdf.