mUOV: Masking the Unbalanced Oil and Vinegar Digital Sigital Signature Scheme at First- and Higher-Order

Suparna Kundu*
COSIC, KU Leuven
Leuven, Belgium
suparna.kundu@esat.kuleuven.be

Quinten Norga*
COSIC, KU Leuven
Leuven, Belgium
quinten.norga@esat.kuleuven.be

Uttam Kumar Ojha*
Indian Statistical Institute
Kolkata, India
uttamkumarojha1729@gmail.com

Anindya Ganguly
Indian Institute of Technology
Kanpur, India
anindyag@cse.iitk.ac.in

Angshuman Karmakar
Indian Institute of Technology
Kanpur, India
angshuman@cse.iitk.ac.in

Ingrid Verbauwhede
COSIC, KU Leuven
Leuven, Belgium
ingrid.verbauwhede@esat.kuleuven.be

ABSTRACT

The National Institute for Standards and Technology (NIST) initiated a standardization procedure for additional digital signatures and recently announced round-2 candidates for the PQ additional digital signature schemes. The multivariate digital signature scheme Unbalanced Oil and Vinegar (UOV) is one of the oldest post-quantum schemes and has been selected by NIST for Round 2. Although UOV is mathematically secure, several side-channel attacks (SCA) have been shown on the UOV or UOV-based digital signatures. We carefully analyze the sensitivity of variables and operations in the UOV scheme from the side-channel perspective and show which require protection. To mitigate implementation-based SCA, we integrate a provably secure arbitrary-order masking technique with the key generation and signature generation algorithms of UOV. We propose efficient techniques for the masked dot-product and matrix-vector operations, which are both critical in multivariate DS schemes. We also implemented and demonstrate the practical feasibility of our masking algorithms for UOV-Ip on the ARM Cortex-M4 microcontroller. Our first-order masked UOV implementations have 2.7× and 3.6× performance overhead compared to the unmasked scheme for key generation and signature generation algorithms. Our first-order masked UOV implementations use 1.3× and 1.9× stack memory rather than the unmasked version of the key generation and signature generation algorithms.

KEYWORDS

Post-Quantum Cryptography, Masking, Multivariate-based Digital Signatures, UOV.

1 INTRODUCTION

The National Institute for Standards and Technology (NIST) selected its first Post-Quantum (PQ) Digital Signature (DS) standards in 2022 [1] and published them recently [26, 27]. The current PQ DS standards of NIST are (i) Dilithium [15, 26], (ii) Falcon [17], and (iii) Sphincs+ [7, 27]. Two of the three standard DS schemes ((i) and (ii)) are based on structural lattices. Therefore, a crypt-analytical attack on the structural lattice-based hard problem may endanger all the PQ standards. To prevent such disastrous situations and broaden its portfolio by including DS schemes based on different hard mathematical problems, NIST initiated another standardization procedure for additional digital signatures [25]. In this call, NIST has shown interest in DS schemes with short signatures and fast verifications. Recently, NIST announced round-2 candidates for the PQ additional digital signature schemes [28].

The Unbalanced Oil and Vinegar (UOV) is a multivariate DS scheme. It follows the hash-and-sign paradigm, which uses multivariate quadratic polynomial maps as trapdoor functions. This scheme was first proposed by Kipnis et al. [24] in 1999 and later modified by Beullens et al. and submitted to the additional digital signatures standardization procedure of NIST [9]. This scheme has shorter signatures compared to DS standards. This scheme has also advanced to the second round of NIST's competition and is a potential candidate for additional DS standards. Interestingly, three other candidates from the second round, QR-UOV [18], SNOVA [35], and Mayo [8], also use the UOV principle. A second-round candidate in the ongoing Korean PQC standardization procedure [32], MQ-Sign [34] is also based on the UOV principle.

UOV is one of the oldest PQ schemes, so its crypt-analytical security is time-tested. However, several side-channel attacks (SCA) have been demonstrated on the UOV DS scheme [2, 3, 30, 36]. These attacks exploit power consumption information (electromagnetic radiation) leakages from the physical device while executing the cryptographic algorithms. Therefore, integration of the countermeasure with the implementations of the UOV scheme is essential to prevent these attacks.

1.1 Contribution

The key generation algorithm generates the secret key, and the signature generation uses this non-ephemeral secret key to generate signatures. Key generation and signature generation are the two algorithms of the UOV scheme that are vulnerable to SCA. We perform a thorough senstivity analysis on these procedures, identifying the operations and variables which require side-channel protection. Masking is a widely known provable secure countermeasure against SCA. In this work, we propose provably secure arbitrary-order masked algorithms for key generation, secret key expansion, and signature generation operations. To this end, we propose several novel gadgets, including the `SecDotProd` gadget. Our approach allows to efficiently compute the masked cross-products of all vector elements and delaying share re-masking and final compression. The proposed *lazy compression* is more efficient as it does not require re-sharing and compression after each coefficient multiplication, and

*Three authors contributed equally to this research.

allows us to construct efficient and secure matrix-vector multiplications. Further, we would like to note that our approach is not limited to the UOV scheme, it can be extended to other UOV-based multivariate schemes, such as Mayo, QR-UOV, SNOVA, and MQ-Sign.

Finally, we implemented our masking algorithms for UOV-Ip on the ARM Cortex-M4 microcontroller. We include the implementation results in this paper. Our first-order masked UOV implementations have 2.7× and 3.6× performance overhead compared to the unmasked scheme for key generation and signature generation algorithms, respectively. Our first-order masked UOV implementations use 1.3× and 1.9× stack memory with respect to the unmasked version of the key generation and signature generation algorithms, respectively.

2 PRELIMINARIES

2.1 Notation

We use $\mathbb{F}_q$ to denote a finite field with q elements and q a power-of-two positive integer. All vectors and matrices are defined over $\mathbb{F}_q$. Lower-case letters (e.g., x) denote field elements/ coefficients, lower-case bold letters (e.g., $\mathbf{v}$) represent vectors and upper-case bold letters denote matrices (e.g., $\mathbf{M}$). All vectors are in the column form, and the transpose of the matrix $\mathbf{M}$ is denoted by $\mathbf{M}^\top$. The identity matrix of size m is denoted by $\mathbf{I}_m$, while $\mathbf{0}_k$ is the zero column vector. $x \leftarrow S$ represents the (random) sampling of x from the set S. The ith bit position of a field element x is represented with $x^{[i]}$. The jth element of the vector $\mathbf{v}$ is indicated as $\mathbf{v}[j]$. The (j,k)th element of the matrix $\mathbf{M}$ is represented as $\mathbf{M}[j,k]$ and the elements of the positions (j,k) to $(j,k+l)$ of the matrix $\mathbf{M}$ is represented collectively as $\mathbf{M}[j,k{:}k+l]$. A sequence of n shares $(x_1,\ldots,x_n)$ of a sensitive variable x is represented as $(x_i)_{1\le i\le n}$ or (x_i), when the number of shares n is clear from context.

2.2 The UOV digital signature scheme

Throughout this work, we target the NIST-submitted version of the UOV DS scheme [9], which we now briefly introduce.

The *UOV map* $\mathcal{P}:\mathbb{F}_q^n \to \mathbb{F}_q^m$ is a multivariate quadratic map that vanishes on a m-dimensional secret linear subspace O. We represent the matrix form of this quadratic map as $\mathcal{P}=(\mathbf{P}_1,\cdots,\mathbf{P}_m)$. Each matrix $\{\mathbf{P}_i\}_{i\in[m]}$ can be represented by three block matrices: $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(3)}$ are two upper triangular square block matrices of size $(n-m)$ and m respectively; and $\mathbf{P}_i^{(2)}$ is a matrix of size $(n-m)\times m$. Interestingly, $\mathbf{P}_i^{(1)},\mathbf{P}_i^{(2)},\mathbf{P}_i^{(3)}$ and $\mathbf{O}$ are related through the following Equation 1.

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

We present the UOV digital signature scheme in Fig. 1, which consists of three main algorithms: (i) compact key generation, (ii) signature generation, and (iii) verification.

(i) CompactKeyGen(): the compact key generation algorithm first samples two seeds $\mathrm{seed_{sk}}$ and $\mathrm{seed_{pk}}$. The $\mathrm{seed_{sk}}$ is used as input in $\mathsf{Expand_{sk}}$ to generate the secret matrix $\mathbf{O}$, which corresponds to the oil space. Then, the $\mathrm{seed_{pk}}$ is used as input in $\mathsf{Expand_{pk}}$ to construct two public matrices: $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(2)}$. Further, $\mathbf{P}_i^{(3)}$ is computed following Eq. 1. Finally, the compressed public key cpk and secret key csk are returned. The compressed csk and cpk can be expanded to esk and epk using the ExpandSK and ExpandPK routines (Figure 2).

`CompactKeyGen()`

$\mathrm{seed_{sk}} \leftarrow \{0,1\}^{\mathrm{sk\_seed\_len}}$
$\mathrm{seed_{pk}} \leftarrow \{0,1\}^{\mathrm{pk\_seed\_len}}$
$\mathbf{O} := \mathsf{Expand_{sk}}(\mathrm{seed_{sk}})$ ## Figure 2
$\left\{\mathbf{P}_i^{(1)},\mathbf{P}_i^{(2)}\right\}_{i\in[m]} := \mathsf{Expand_{P}}(\mathrm{seed_{pk}})$ ## Figure 2
for $i=1$ upto m **do**
    $\mathbf{P}_i^{(3)} := \mathsf{Upper}\left(-\mathbf{O}^\top\mathbf{P}_i^{(1)}\mathbf{O}-\mathbf{O}^\top\mathbf{P}_i^{(2)}\right)$
$cpk := \left(\mathrm{seed_{pk}},\left\{\mathbf{P}_i^{(3)}\right\}_{i\in[m]}\right)$
$csk := (\mathrm{seed_{pk}},\mathrm{seed_{sk}})$
return (cpk,csk)

`Sign`(esk,μ)

$\mathrm{salt} \leftarrow \{0,1\}^{\mathrm{salt\_len}}$
$\mathbf{t} := \mathsf{Hash}(\mu||\mathrm{salt})$
for $ctr=0$ upto 255 **do**
    $\mathbf{v} := \mathsf{Expand_v}(\mu||\mathrm{salt}||\mathrm{seed_{sk}}||\mathrm{ctr})$
    $\mathbf{L} := \mathbf{0}_{m\times m}$
    for $i=1$ upto m **do**
        Set i-th row of $\mathbf{L}$ to $\mathbf{v}^\top\mathbf{S}_i$
    $\mathbf{y} := [\mathbf{v}^\top\mathbf{P}_i^{(1)}\mathbf{v}]_{i\in[m]}$
    $\mathbf{x} := \mathbf{L}^{-1}(\mathbf{t}-\mathbf{y})$ ## $\mathbf{x}=\perp$ if $\det(\mathbf{L})=0$
    if $\mathbf{x}\neq\perp$ **then**
        $\mathbf{s} := \begin{bmatrix}\mathbf{v}\\ \mathbf{0}_m\end{bmatrix}+\begin{bmatrix}\mathbf{O}\\ \mathbf{I}_m\end{bmatrix}\mathbf{x}$
        $\sigma := (\mathbf{s},\mathrm{salt})$
        return σ
return $\perp$

`Verify`(epk,μ,σ)

$\mathbf{t} := \mathsf{Hash}(\mu||\mathrm{salt})$
return $\mathbf{t} == [\mathbf{s}^\top\mathbf{P}_i\mathbf{s}]_{i\in[m]}$

Figure 1: UOV DS scheme [9]

(ii) Sign(esk,μ): the signature generation algorithm takes the secret key esk and message μ as input. It first samples a random salt, and computes the message digest $\mathbf{t}$ as $\mathsf{Hash}(\mu||\mathrm{salt})$. We then compute the pre-image $\mathbf{s}$ of $\mathbf{t}$ using the secret key via rejection sampling. This requires uniformly sampling a vinegar vector $\mathbf{v}\in\mathbb{F}_q^n$ and then computing $\mathbf{y} = [\mathbf{v}^\top\mathbf{P}_i^{(1)}\mathbf{v}]_{i\in[m]}$. Therefore, the quadratic system $\mathcal{P}(\mathbf{s})=\mathbf{t}$ is converted to a linear system $\mathbf{L}\mathbf{x}=\mathbf{t}-\mathbf{y}$. If $\mathbf{L}$ is invertible, then $\mathbf{x}$ can be computed by performing Gaussian elimination, allowing the computation of $\mathbf{s}$, finally. Otherwise, $\mathbf{v}$ is re-sampled and the previous process is repeated. The final signature σ consists of $\mathbf{s}$ and salt.

(iii) Verify(epk, μ, σ): the verification algorithm takes public key epk, message μ, and signature σ as input. At first, it compute the message digest $\mathbf{t}$ as $\mathsf{Hash}(\mu||\mathrm{salt})$. Then, it evaluates the UOV map as $\mathcal{P}(\mathbf{s})=(\mathbf{s}^\top\mathbf{P}_1\mathbf{s},\cdots,\mathbf{s}^\top\mathbf{P}_m\mathbf{s})$ and checks whether it matches with $\mathbf{t}$. Based on this, it outputs accept or reject.

We present the parameter set of the different variants of UOV in Table 1. For all the versions of UOV, `pk_seed_len` = 128, `sk_seed_len` = 256 & `salt_len` = 128. Throughout this text we will denote vector/matrix dimension $n-m$ as l.

ExpandSK(csk)

(1) $\mathbf{O} := \mathsf{Expand}_{\mathsf{sk}}(\mathsf{seed}_{\mathsf{sk}})$

(2) $\left\{\mathbf{P}_i^{(1)},\mathbf{P}_i^{(2)}\right\}_{i\in[m]} := \mathsf{Expand}_{\mathbf{P}}(\mathsf{seed}_{\mathsf{pk}})$

(3) **for** $i=1$ upto m **do**

(4) $\quad \mathbf{S}_i := \left(\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top}\right)\mathbf{O} + \mathbf{P}_i^{(2)}$

(5) $esk := \left(\mathsf{seed}_{\mathsf{sk}}, \mathbf{O}, \left\{\mathbf{P}_i^{(1)}, \mathbf{S}_i\right\}_{i\in[m]}\right)$

(6) **return** esk

ExpandPK(cpk)

(1) $\left\{\mathbf{P}_i^{(1)},\mathbf{P}_i^{(2)}\right\}_{i\in[m]} := \mathsf{Expand}_{\mathbf{P}}(\mathsf{seed}_{\mathsf{pk}})$

(2) **for** $i=1$ upto m **do**

(3) $\quad \mathbf{P}_i = \begin{bmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0} & \mathbf{P}_i^{(3)} \end{bmatrix}$

(4) $epk := \{\mathbf{P}_i\}_{i\in[m]}$

(5) **return** epk

Figure 2: Expand functions of the UOV DS scheme [9]

Table 1: Parameter sets of different versions of UOV

Scheme	Security Level	Parameters		
		n	m	q
UOV-Ip	I	112	44	256
UOV-Is		160	64	16
UOV-III	III	184	72	256
UOV-V	V	244	96	256

2.3 Masking

Introduced by Chari et al. [10], masking is a popular countermeasure to protect against SCA attacks. The fundamental idea is to split sensitive variables x into several randomized shares $(x_1,\dots,x_n)$ so that an attacker needs to learn something about all shares to learn about the original secret x. In this work, we use Boolean masking, where $x = x_1 + \dots + x_n$, and the addition is a logical XOR ($\oplus$).

Ishai et al. [21] introduced the t-probing model, a theoretical framework to argue about the practical security of the masking countermeasure. It allows an adversary to probe t intermediate values in a masked implementation: if any such t probes do not leak information about the unshared secret, the implementation is t-probing secure. As a masked algorithm or circuit grows in size, it becomes increasingly complex to analyze its security. A solution is to split the large function into smaller *gadgets* and prove their security. Barthe et al. [4] introduced several security notions, which allow us to prove the probing security of such gadget compositions. We recall these security notions, as presented in [33].

Definition 2.1 (t-(Strong-)Non-Interference (t-(S)NI) security). A gadget with one output sharing and m_i input shares is t-NI (resp. t-SNI) secure if any set of at most t_1 probes on its internal wires and t_2 probes on wires from its output sharings such that $t_1 + t_2 \le t$ can be simulated with $t_1 + t_2$ (resp. t_1) shares of each of its m_i input sharings.

We also recall two extensions for these notions, which are required when masking digital signature schemes. These involve making values public, such as the computed signatures. More specifically, all outputs of a free-t-SNI gadget can be simulated using all but one of its input shares and the unmasked output [14]. The t-NIo notion [5] gives the simulator access to certain intermediate values to ensure successful simulation.

Definition 2.2 (free-t-Strong-Non-Interference (free-t-SNI) security). A gadget with one output sharing b_i and m_i input sharings is free-t-SNI secure if any set of at most t_1 probes on its internal wires such that $t_1 \le t$ there exists a subset I of input indices with $|I| \le t_1$, such that the t_1 intermediate variables and the output variables $b_{|I}$ can be perfectly simulated from $a_{|I}$, while for any $O \subsetneq [1,n] \setminus I$ the output variables in $c_{|O}$ are uniformly and independently distributed, conditioned on the probed variables and $c_{|I}$.

Definition 2.3 (t-Non-Interference with public outputs (t-NIo) security). A gadget with public output b and m_i input sharings is t-NIo secure if, for any set of $t_1 \le t$ intermediate variables, there exists a subset I of input indices with $|I| \le t_1$, such that t_1 intermediate variables can be perfectly simulated from $x_{|I}$ and b.

3 SENSITIVITY ANALYSIS

In this section, we analyze the sensitivity of all the variables and operations of the UOV scheme. The goal is to identify the sensitive variables which need to be protected against side-channel leakage and Differential Power Attacks (DPA). We draw the sensitive UOV procedures in color-coded diagrams, Figure 3 - 5. All public data, including (compact/expanded) public key, message and signature of a message, are non-sensitive and indicated in blue. All sensitive data, and operations dealing with them, are highlighted in red.

First of all, since the ExpandPK(cpk) and Verify(epk,μ,σ) do not manipulate sensitive data, those algorithms are non-sensitive and do not require masking. The remaining algorithms, i.e., compact key generation, secret key expansion, and signature generation, process sensitive data. Note that $\mathsf{seed}_{\mathsf{pk}}$ and $\left\{\mathbf{P}_i^{(1)}\right\}_{i\in[m]}$ in csk and esk, respectively, are also part of the public key, and thus are non-sensitive.

3.1 Compact Key Generation and Secret Key Expansion

During key generation and secret key expansion, $\mathsf{seed}_{\mathsf{sk}}$ is sensitive as it is the seed for $\mathbf{O}$ and subsequent values. Additionally, $\left\{\mathbf{P}_i^{(3)}\right\}_{i\in[m]}$ has to be protected during the computation, as $\mathbf{O}$ is involved. The final value is revealed as part of the public key cpk. In contrast, $\{\mathbf{S}_i\}_{i\in[m]}$ is sensitive as it is also derived from $\mathbf{O}$ and requires protection as part of the secret key esk. All other variables in the key generation and expansion can be leaked or are public and do not require side-channel protection.

3.2 Signature Generation

In the signature generation, $\mathbf{t}$ is the hash of a message μ and salt is part of the signature σ, so both are non-sensitive. Pébereau [31] shows one vector from secret oil space is enough for a key recovery attack. If $\mathbf{v}$ is known, $\mathbf{s} - \begin{bmatrix}\mathbf{v}\\ \mathbf{0}_m\end{bmatrix}$, an oil vector, is also known. So, $\mathbf{v}$ is sensitive. If $\mathbf{L}$ is known, $\mathbf{y} = \mathbf{t} - \mathbf{L}\mathbf{x}$ is also known and thus requires side-channel protection. Due to the structure of the secret oil space, $\mathbf{x}$ are the linear coefficients of the oil vector and are part of the signature $\mathbf{s}$. Thus, $\mathbf{x}$ can

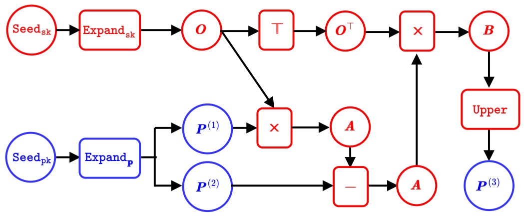

Figure 3: Sensitivity analysis of `CompactKeyGen()` = (cpk, csk).

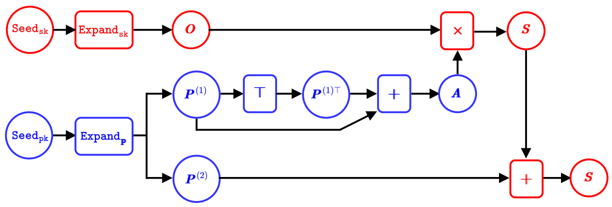

Figure 4: Sensitivity analysis of `ExpandSK`(csk) = (esk).

be revealed after computation. The execution time of signature generation leaks *ctr* value, so we can consider *ctr* is also non-sensitive.

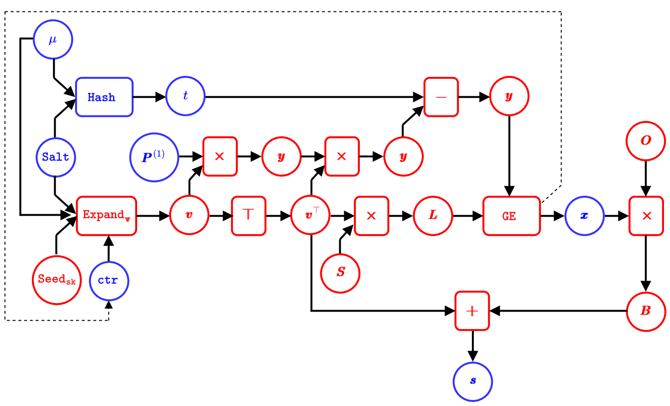

Figure 5: Sensitivity analysis of `Sign`(esk, μ) = (σ).

4 MASKING UOV AT ARBITRARY ORDER

We now introduce and describe the complete first- and high-order masking of UOV, a NIST Additional DS Round 1 candidate. All novel gadgets are described by a t-order algorithm ($n = t+1$ shares) and accompanied with a detailed description. The main algorithms are masked key generation (`mCompactKeyGen`, Alg. 4), secret key expansion (`mExpandSK`, Alg. 5) and signing (`mSign`, Alg. 6). As the signature verification procedure operates only on public values, no masking is required.

First, in Section 4.1 - 4.4, we introduce several novel masked gadgets which are used as subroutines in the main algorithms, including:

- `SecDotProd` and `SecMatVec`: efficient masked dot product on two Boolean masked vectors. It is the main building block for matrix-vector multiplication, as used during key generation and signing.
- `SecQuad`: masked evaluation of a quadratic form, as used during signing.

All components (Table 2), including gadgets from literature, are put together to achieve fully masked UOV in Section 4.5 - 4.7.

Table 2: Overview of used gadgets in this work, with $n = t+1$ shares.

Algorithm	Description	Security	Reference
`SecREF`	Refresh of Boolean masking	t-SNI	[4, 12]
`FullAdd`	Secure unmasking of Boolean shares	t-NI	[5, 11] & Alg. 7
`SecDotProd`	Dot prod. of two Boolean masked vectors	t-SNI	Algorithm 1
`SecMatVec`	Matrix-vector multiplication	t-SNI	Algorithm 2
`SecQuad`	Evaluation of a quadratic form	t-SNI	Algorithm 3
`SecRowEch`	Matrix conversion to row echelon form	t-NIo	[29] & Alg. 8
`SecBackSub`	Masked back substitution with public output	t-NIo	[29] & Alg. 9
`mCompactKeyGen`	Masked UOV Compact Key Generation	t-NIo	Algorithm 4
`mExpandSK`	Masked UOV Secret Key Expansion	t-NI	Algorithm 5
`mSign`	Masked UOV Signature Generation	t-NIo	Algorithm 6

Methodology. We prove all algorithms/gadgets to be t-(S)NI secure in the probing model via simulation. We show how probes on intermediate variables and output shares of a gadget can be perfectly simulated with only a limited number of input shares. For algorithms which are composed from multiple gadgets, we rely on the t-(S)NI properties of the sub-gadgets to argue about simulatability of all values. For example, the set of probes required from the input shares of a t-SNI gadget is independent from the amount of probes on its output shares. By iterating over all possible intermediate (and output) variables of each sub-gadget, starting at the output and moving to the input of the algorithm, all required probes for simulation are summed.

4.1 Masked Dot Product

The (masked) matrix-vector multiplication operation is critical in multivariate-based post-quantum crypto. As highlighted in Section 2, it is also the case for the UOV scheme. We propose a method to efficiently compute the masked dot product (`SecDotProd`) using *lazy compression*. The computation of a masked multiplication involves three stages: computation of cross-products, re-sharing and compression into the final n shares. Computing a dot-product of two l-dimensional vectors requires performing l masked multiplications and summing them. By delaying the re-sharing and compression of the cross-products, until completing them for all l elements in the input vectors $\mathbf{x}$ and $\mathbf{y}$, we only need to perform them once at the end. We now discuss our approach in detail, which is inspired by the approach in [19], modifying the domain-oriented ISW multiplication [20, 21] by delaying the compression stage when chaining multiplications.

Computation of l cross-products. The cross-products for l input coefficients of $(\mathbf{x}_i)$ and $(\mathbf{y}_i)$ are computed and summed. We observe here that since no cross-products are combined, and all input coefficients are independent, they can be computed independently and each summed together.

Resharing. The cross-products which contain shares of both inputs with different share indices ($i \neq j$) are now refreshed using a fresh random share. This is to prevent the re-combination of all shares of a single coefficient in the following step.

Algorithm 1: SecDotProd

Data: Boolean sharings $(\mathbf{x}_i)$ and $(\mathbf{y}_i)$ of vectors $\mathbf{x},\mathbf{y}\in\mathbb{F}_q^l$.
Result: A Boolean sharing (z_i) of a coefficient $z=\mathbf{x}^T\mathbf{y}\in\mathbb{F}_q$.

$(u_{ij}),(w_i) := 0$
Compute and sum l cross-products
for $k=1$ *upto* l **do**
  for $i=1$ *upto* n **do**
    for $j=i+1$ *upto* n **do**
      $u_{ij}=u_{ij}+\mathbf{x}[k]_i\mathbf{y}[k]_j$
      $u_{ji}=u_{ji}+\mathbf{x}[k]_j\mathbf{y}[k]_i$
  $(w_i)=(w_i+\mathbf{x}[k]_i\mathbf{y}[k]_i)$
Resharing
for $i=1$ *upto* n **do**
  for $j=i+1$ *upto* n **do**
    $r_{ij}\leftarrow\mathbb{F}_q$
    $u_{ij}=u_{ij}+r_{ij}$
    $u_{ji}=u_{ji}+r_{ij}$
Compression
$(z_i):=(w_i+\sum_{j=1,j\neq i}^{n}u_{ij})$
return (z_i)

Compression. The refreshed partial sums are now combined into the final output values z_i. As proposed in [16], it is critical (for security) that the result of the computation of z_i is stored in a memory element and only the full result is returned. This is not necessary for probing security, but required for t-SNI security. It is clear that only performing the re-sharing and compression step once, as proposed here, is more efficient than performing it for every input coefficient pair and summing the results of those multiplications.

4.1.1 Complexity. The run-time and randomness complexity of SecDotProd are:

$$T_{\texttt{SecDotProd}}(l,n)=l\cdot n\cdot\left(\frac{2n(n-1)}{2}+1\right)+n\cdot\frac{3n(n-1)}{2}+n(n-1)$$
$$=ln^3-ln^2+ln+\frac{3}{2}n^3-\frac{1}{2}n^2-n,$$
$$R_{\texttt{SecDotProd}}(l,n,w)=n\cdot\frac{n(n-1)}{2}\cdot w=\frac{1}{2}n^3w-\frac{1}{2}n^2w.$$

4.1.2 Security. We now show that the SecDotProd gadget is t-SNI secure with $n=t+1$ shares, providing resistance against a probing adversary with t probes and allowing us to use the gadget in larger compositions.

LEMMA 4.1. *The gadget* SecDotProd *(Algorithm 1) is t-SNI secure.*

Proof. The full proof is included in Appendix A.

4.2 Masked Matrix-Vector Multiplication

We now show how the optimized SecDotProd gadget is used to compute a masked matrix vector multiplication (SecMatVec) in an efficient manner. As shown in Algorithm 2, by applying the dot product on each row (m in total) of a Boolean masked matrix $(\mathbf{A}_i)$, the shared vector $(\mathbf{b}_i)$ with $\mathbf{b}=\mathbf{A}\mathbf{x}\in\mathbb{F}_q^m$ can be computed (m iterations, m coefficients).

Algorithm 2: SecMatVec

Data: 1. A Boolean sharing $(\mathbf{A}_i)$ of a matrix $\mathbf{A}\in\mathbb{F}_q^{m\times l}$.
2. A Boolean sharing $(\mathbf{x}_i)$ of a vector $\mathbf{x}\in\mathbb{F}_q^l$.
Result: A Boolean sharing $(\mathbf{b}_i)$ of the vector $\mathbf{b}=\mathbf{A}\mathbf{x}\in\mathbb{F}_q^m$

for $j=1$ *upto* m **do**
  $(\mathbf{b}[j]_i):=$ SecDotProd$((\mathbf{A}[j,:]_i),(\mathbf{x}_i))$
return $(\mathbf{b}_i)$

4.2.1 Complexity & Security. The run-time and randomness complexity of SecMatVec are:

$$T_{\texttt{SecMatVec}}(l,m,n)=m\cdot T_{\texttt{SecDotProd}}(n,l)$$
$$=lmn^3-lmn^2+lmn+\frac{3}{2}mn^3-\frac{1}{2}mn^2-mn,$$
$$R_{\texttt{SecMatVec}}(l,m,n,w)=m\cdot R_{\texttt{SecDotProd}}(n,l,w)$$
$$=\frac{1}{2}mn^3w-\frac{1}{2}mn^2w.$$

We now prove Algorithm 2 to be t-SNI secure with $n=t+1$ shares, providing resistance against a probing adversary with t probes and allowing us to use the gadget in larger compositions.

LEMMA 4.2. *The gadget* SecMatVec *(Algorithm 2) is t-SNI secure.*

Proof. This is a direct result from the SecDotProd gadget being t-SNI secure. As each iteration is t-SNI secure and independent, the whole loop is t-SNI too. It is clear that if an adversary can probe t times in total across different iterations or independent outputs, these can be simulated with no more number of input shares. □

4.3 Masked Quadratic Form Evaluation

The quadratic form evaluation is used in the UOV scheme to compute the vector $\mathbf{y}=[\mathbf{x}^T\mathbf{P}_j\mathbf{x}]_{j\in[m]}$. Our masked gadget operates on the Boolean shares $(\mathbf{x}_i)$ and public matrices $\{\mathbf{P}_j\}_{j\in[m]}$, and it is described in Algorithm 3. The computation happens in two steps: first the masked matrix $(\mathbf{T}_i)=(\mathbf{P}_j\mathbf{x}_i)$ is computed in a share-wise manner, using m public matrices to compute its m columns. After which the SecMatVec gadget is used to compute the matrix-vector multiplication $(\mathbf{y}_i)=(\mathbf{x}_i^T)(\mathbf{T}_i)$ on two Boolean shared operands.
Computation of T= $\{\mathbf{P}_j\}_{j\in[m]}\mathbf{x}$**.** As the m matrices $\{\mathbf{P}_j\}$ are public, they can be multiplied in a share-wise manner with the sensitive vector $(\mathbf{x}_i)$. Each masked multiplication (Line 3) is a column of matrix $(\mathbf{T}_i)$.
Computation of $\mathbf{y}=\mathbf{x}^T\mathbf{T}$**.** After the full Boolean masked matrix $(\mathbf{T}_i)$ is constructed, it is multiplied with Boolean masked $(\mathbf{x}_i)$ on Line 4. Here, we rely on the property $(\mathbf{x}^T\mathbf{T})^T=\mathbf{T}^T\mathbf{x}$ to calculate the desired result through the SecMatVec gadget. Also, the masking of vector $(\mathbf{x}_i)$ is first refreshed to ensure both inputs of the gadget are independent (Line 1).

Algorithm 3: SecQuad

Data: 1. Public matrices $\{\mathbf{P}_j \in \mathbb{F}_q^{l\times l}\}_{j\in[m]}$.
2. A Boolean sharing $(\mathbf{x}_i)$ of the vector $\mathbf{x} \in \mathbb{F}_q^l$

Result: A Boolean sharing $(\mathbf{y}_i)$ of the vector $\mathbf{y} = [\mathbf{x}^T \mathbf{P}_j \mathbf{x}]_{j\in[m]} \in \mathbb{F}_q^m$

```
1 (s_i) := SecREF((x_i))
2 for j = 1 upto m do
3     (T[:,j]_i) = (P_j x_i)                 /* T_i ∈ F_q^{l×m} */
4 (y_i) := SecMatVec((T_i^T),(s_i))   /* y^T = (x^T T)^T = T^T x */
5 return (y_i)
```

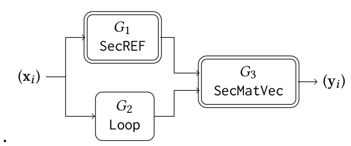

Figure 6: An abstract diagram of SecQuad (Algorithm 3). The t-NI gadgets are depicted with a single border, the t-SNI gadgets with a double border.

4.3.1 Complexity. The run-time and randomness complexity of SecQuad are:

$$\begin{aligned}T_{\mathsf{SecQuad}}(l,m,n) &= T_{\mathsf{SecREF}}(n,l) + m\cdot l^2 \cdot n + T_{\mathsf{SecMatVec}}(n,l,m)\\ &= (\frac{3}{2}ln^2 - \frac{3}{2}ln) + (\frac{1}{2}l^2m^2n + \frac{1}{2}l^2mn)\\ &\quad + (lmn^3 - lmn^2 + lmn + \frac{3}{2}mn^3 - \frac{1}{2}mn^2 - mn),\\ R_{\mathsf{SecQuad}}(l,m,n,w) &= R_{\mathsf{SecREF}}(n,l,w) + R_{\mathsf{SecMatVec}}(n,l,m,w)\\ &= (\frac{1}{2}ln^2w + \frac{1}{2}lnw) + (\frac{1}{2}mn^3w - \frac{1}{2}mn^2w).\end{aligned}$$

4.3.2 Security. We now argue about the first- and high-order security of Algorithm 3 by proving it to be t-SNI secure with $n = t+1$ shares. This means it provides resistance against an adversary with t probes and allows using the algorithm in larger compositions.

Lemma 4.3. *The gadget* SecQuad *(Algorithm 3) is t-SNI secure.*

Proof. Figure 6 depicts an overview of the construction of Algorithm 3 from its elementary gadgets. Apart from those listed in Table 2, we model the loop of linear operations in Line 2-3 as a t-NI gadget G_2 ('Loop'), which we prove first. Subsequently, we prove the security of the larger composition.

We first argue that a single iteration (Line 3) is t-NI, which is trivial as the inputs are processed in a share-wise manner. Similar as before, if an attacker can probe across different independent iterations, the t intermediate values can be simulated with no more number of shares of input $(\mathbf{x}_i)$. As a result, the whole loop is considered to be executed in parallel and modeled as single t-NI gadget G_2.

We now prove that the combination of all operations (whole gadget) are t-SNI (Lemma 4.3). An adversary can probe each gadget (G_i) internally or at its output. The number of internal and output probes for each gadget are denoted as t_{G_i} and o_{G_i}, respectively. The total number of probes t_{A_3} and output shares $|O|$ of Algorithm 3 are:

$$t_{A_3} = \sum_{i=1}^{3} t_{G_i} + \sum_{i=1}^{2} o_{G_i}, \quad |O| = o_{G_3}$$

We show that the internal and output probes can be perfectly simulated with $\leq t_{A_3}$ input shares. Firstly, to simulate the internal and output probes on gadget G_3, only t_{G_3} shares of both inputs are required. This is a direct result of the t-SNI property of G_3: the simulation of a t-SNI gadget can be performed independent of the number of probed output shares. As a direct result, the propagation of output shares to the input shares is stopped. The simulation succeeds on a column-level as G_2 produces m independent outputs and t_{G_3} shares of m independent columns are required Secondly, the simulation of t_{G_2} internal and o_{G_2} output probes on gadget G_2 requires $t_{G_2} + o_{G_2}$ shares of its input, as it is t-NI. Finally, due to the t-SNI property of gadget G_1, t_{G_1} input shares are required to simulate t_{G_1} intermediate probes and o_{G_1} output shares. Finally, we sum up the required shares of the inputs for simulation of all gadgets $|I|$. As $|I| = t_{G_1} + t_{G_2} + o_{G_2} + t_{G_3} \leq t_{A_3}$ and independent from $|O|$, Algorithm 3 is t-SNI. □

4.4 Other Auxiliary Gadgets

4.4.1 FullAdd *(Alg. 7).* For securely unmasking sensitive values and making them public, e.g. the signature after signing, we rely on the FullAdd gadget. Its two main steps are a *strong* (free-t-SNI) mask refreshing and combining all shares. The free-t-SNI notion allows for the simulation of all outputs of the refresh (y_i) with all but one share of the input (x_i), and the unmasked value y [12]. As a result, the subsequent unmasking (which involves all shares) can be perfectly simulated. In contrast, standard t-(S)NI refresh would result in unsound simulation as all shares of its input would be required, which is not probing secure. It is shown in [12] that the t-SNI refresh in [4] also satisfies the free-t-SNI notion. We refer to [12, 13] for the security proof of its t-NIo with public output y notion.

4.4.2 SecRowEch *&* SecBackSub *(Alg. 8 & 9).* A method for solving a masked system of linear equations using (masked) Gaussian elimination with back substitution was proposed in [29]. We recall the SecRowEch and SecBackSub gadgets in Appendix B. Their approach relies on converting a shared matrix $(\mathbf{T}_i)$ to its row-echelon representation by making leading pivot-elements 1. If the matrix is invertible, and thus has a unique solution x, it can be found by performing back substitution on the reduced matrix. We refer to the original work for the complexity and security analysis, including their t-NIo security proofs.

4.5 Masked UOV (Compact) Key Generation

The compact key generation of UOV is used to generate the compact public key *cpk* and compact secret key *csk*. Our approach consists of splitting secret key components and derived (ephemeral) secrets into multiple shares and performing their operations in a masked fashion. Our masking strategy is formally described in Algorithm 4 and shown in Figure 7.

When masked, the compact secret key *csk* is defined as $(seed_{pk}, (seed_{sk,i})_{1\leq i\leq n})$ with the secret-key component seed_{sk} returned as a Boolean sharing. Each share is a randomly sampled binary string of length sk_seed_len. Both compact secret key components are used to compute the upper-triangluar matrix $\mathbf{P}_j^{(3)}$, which is unmasked after computation and returned as part of the compact public key $cpk = (seed_{pk}, \{\mathbf{P}_j^{(3)}\}_{j\in[m]})$. This procedure is explained below.

Algorithm 4: `mCompactKeyGen`

Result: Compact public key and Boolean shared compact secret key (cpk, csk)

1 $\mathrm{seed_{pk}} \leftarrow \{0,1\}^{\mathrm{pk\_seed\_len}}$
2 $(\mathrm{seed}_{\mathrm{sk},i})_{1\le i\le n} \leftarrow \{0,1\}^{\mathrm{sk\_seed\_len}}$
3 $(\mathbf{O}_i) := \mathsf{mExpand_{sk}}((\mathrm{seed}_{\mathrm{sk},i}))$ /* $\mathsf{O}_i \in \mathbb{F}_q^{l\times m}$ */
4 $\{\mathbf{P}_j^{(1)}, \mathbf{P}_j^{(2)}\}_{j\in[m]} := \mathsf{Expand_{\mathbf{P}}}(\mathrm{seed_{pk}})$ /* $\mathsf{P}_j^{(1)} \in \mathbb{F}_q^{l\times l}$ */
5 $(\mathbf{Q}_i) := \mathsf{SecREF}((\mathbf{O}_i))$ /* $\mathsf{P}_j^{(2)} \in \mathbb{F}_q^{l\times m}$ */
6 **for** $j = 1$ *upto* m **do**
7     $(\mathbf{A}_i) := (-\mathbf{P}_j^{(1)}\mathbf{O}_i)$ /* $\mathrm{A}_i \in \mathbb{F}_q^{l\times m}$ */
8     $\mathbf{A}_1 = \mathbf{A}_1 - \mathbf{P}_j^{(2)}$
9   **for** $k = 1$ *upto* m **do**
10         $(\mathbf{B}[:,k]_i) = \mathsf{SecMatVec}((\mathbf{A}_i^T), (\mathbf{Q}[:,k]_i))$
11     $(\mathbf{C}_i) := \mathsf{Upper}(\mathbf{B}_i)$ /* $\mathrm{C}_i \in \mathbb{F}_q^{m\times m}$ */
12     $\mathbf{P}_j^{(3)} := \mathsf{FullAdd}((\mathbf{C}_i))$
13 **return** $(cpk = (\mathrm{seed_{pk}}, \{\mathbf{P}_j^{(3)}\}_{j\in[m]})$, $csk = (\mathrm{seed_{pk}}, (\mathrm{seed}_{\mathrm{sk},i})_{1\le i\le n}))$

Generation of O. The shares of the secret matrix $\mathbf{O}$ are obtained by expanding the masked seed $((seed_{sk,i})_{1\le i\le n})$ using the masked PRNG $\mathsf{mExpand_{sk}}$ in Line 3. The masked PRNG is instantiated using masked `shake256()`, derived from the Keccak primitive, and produces Boolean shares $(\mathbf{O}_i)$.

Computation of $\{\mathbf{A}_j\}_{j\in[m]} = \{-\mathbf{P}_j^{(1)}\mathbf{O} - \mathbf{P}_j^{(2)}\}_{j\in[m]}$. The m upper-triangular matrices $\mathbf{P}_j$ consist of three sub-matrices. The first two $\{\mathbf{P}_j^{(1)}, \mathbf{P}_j^{(2)}\}_{j\in[m]}$ can be computed in the clear (Line 4), and are used to compute the third $\{\mathbf{P}_j^{(3)}\}_{j\in[m]}$. The first step is to compute m matrices $\{\mathbf{A}_j\}_{j\in[m]}$ in a masked fashion (Line 7). During each of the m iterations, only share-wise (linear) matrix multiplication and subtraction are required. The public matrix $\mathbf{P}_j^{(1)}$ is multiplied with each share of secret matrix $(\mathbf{O}_i)$. As sub-matrix $\mathbf{P}_j^{(2)}$ is also public, it is only subtracted from one (first) share of each $\mathbf{A}_j$ (Line 8).

Computation of $\{\mathbf{B}_j\}_{j\in[m]} = \{\mathbf{O}^T\mathbf{A}_j\}_{j\in[m]}$. The second step is to compute m matrices $\{\mathbf{B}_j\}_{j\in[m]}$ in a masked fashion, which requires multiplying two masked matrices. Each of the resulting m sub-matrices is computed in a column-wise fashion, using our proposed `SecMatVec` gadget. This gadget securely multiplies a shared matrix $(\mathbf{A}_i^T)$ with a shared vector $(\mathbf{Q}[:,k]_i)$ in Line 10, which is column k of a masked matrix $\mathbf{Q}$. The matrix $\mathbf{Q}$ is a full mask refreshing of secret matrix $\mathbf{O}$. We refresh one of the inputs, to ensure both input sharings of the `SecMatVec` gadget are independent.

Recombining the shares of $\{\mathbf{P}_j^{(3)}\}_{j\in[m]}$. The `Upper` function is applied share by share, on each of m matrices $\{\mathbf{B}_j\}$ in Line 11. Finally, one can securely recombine the shares of each $\mathbf{B}_j$ to obtain each $\mathbf{P}_j^{(3)}$, using the `FullAdd` gadget (Line 12). Its details are discussed in Section 4.4 and its security in a larger composition is explained below.

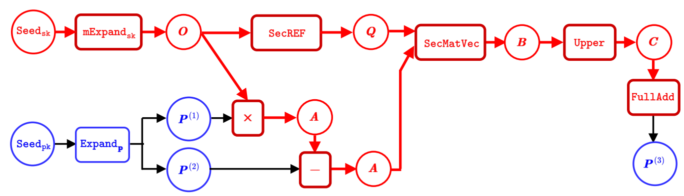

Figure 7: Graphical representation of `mCompactKeyGen()`. Here, red represents masked variables and components, and blue represents unmasked operations and variables. The components introduced or modified due to masking are in dark red color.

4.5.1 *Complexity.*

The run-time and randomness complexity of `mCompactKeyGen` are:

$$\begin{aligned} T_{\mathsf{mCompactKeyGen}}(l,m,n) &= 1 + n + T_{\mathsf{mExpand_{sk}}}(l,m,n) + T_{\mathsf{Expand}_{\boldsymbol{p}}}(l,m,n) \\ &+ T_{\mathsf{SecREF}}(l,m,n) + m\cdot((l^2mn) + (lm) \\ &+ (m\cdot T_{\mathsf{SecMatVec}}(l,m,n)) + (n\cdot T_{\mathsf{Upper}}(m,n)) \\ &+ m^2\cdot T_{\mathsf{FullAdd}}(n)) \\ &= 1 + n + T_{\mathsf{mExpand_{sk}}}(l,m,n) + T_{\mathsf{mExpand_{pk}}}(l,m,n) \\ &+ (\frac{3}{2}lmn^2 - \frac{3}{2}lmn) + (l^2m^2n) + (lm^2) + (lm^3n^3 \\ &- lm^3n^2 + lm^3n + \frac{3}{2}m^3n^3 - \frac{1}{2}m^3n^2 - m^3n) \\ &+ (mn\cdot T_{\mathsf{Upper}}(m,n)) + (\frac{3}{2}m^3n^2 - \frac{3}{2}m^3n + m^3n \\ &- m^3), \end{aligned}$$

$$\begin{aligned} R_{\mathsf{mCompactKeyGen}}(l,m,n,w) &= R_{\mathsf{mExpand_{sk}}}(l,m,n,w) + R_{\mathsf{SecREF}}(l,m,n,w) \\ &+ m\cdot((m\cdot R_{\mathsf{SecMatVec}}(l,m,n,w)) \\ &+ (m^2\cdot R_{\mathsf{FullAdd}}(n,w))) \\ &= R_{\mathsf{mExpand_{sk}}}(l,m,n,w) + (\frac{1}{2}lmn^2w \\ &- \frac{1}{2}lmnw) + (\frac{1}{2}m^3n^3w - \frac{1}{2}m^3n^2w) \\ &+ (\frac{1}{2}m^3n^2w - \frac{1}{2}m^3nw). \end{aligned}$$

4.5.2 *Security.*

To argue about the first- and high-order security of Algorithm 4, we prove it to be t-NIo secure with $n = t+1$ shares and public output $\{\mathbf{P}_j^{(3)}\}_{j\in[m]}$, providing resistance against a probing adversary with t probes. The proof requires us to show how probes on intermediate and output variables in the algorithm can be perfectly simulated with only a limited set of input shares.

Lemma 4.4. *The gadget* `mCompactKeyGen` *(Algorithm 4) is t-NIo secure with public output* $\{\boldsymbol{P}_j^{(3)}\}_{j\in[m]}$.

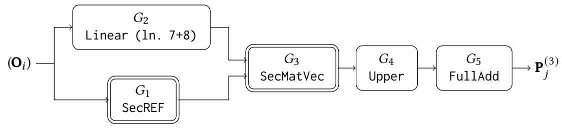

Figure 8: An abstract diagram of an iteration j in `mCompactKeyGen` (Alg. 4). The t-NI gadgets are depicted with a single border, the t-SNI gadgets with a double border.

Proof. We model a single iteration j of Algorithm 4 as a sequence of t-(S)NI gadgets, which is visually shown in Figure 8. In addition to the gadgets listed in Table 2, we model the linear operations in Line 7-8 and Line 11 as t-NI gadgets G_2 and G_4, respectively. This can be trivially shown as the operations are share-wise. Note that the algorithm is independent of the specific masked implementation used for $\mathtt{mExpand}_{\mathsf{sk}}$, which produces a uniformly masked matrix $\mathbf{O}$. We also consider the iterations of the loop in Line 9-10 to be independent and executed in parallel, each generating one of m columns. This means the probes are defined on a column level here (and not variable level) to ensure successful simulation. We summarize the inner loop into a single gadget G_3.

We complete the full proof in two steps: we first prove the composition of gadgets G_1 - G_4 to be t-SNI. Finally, we prove the full Algorithm 4 to be t-NIo, thanks to the final gadget G_5 (`FullAdd`).

Part I: As shown in Figure 8, an adversary can place a number of probes at the output (o_{G_i}) and internally (t_{G_i}) in each gadget G_i. The number of probes of gadget G_1-G_4 of Algorithm 4 are defined as t_{A_4} and output shares $|O|$ with

$$t_{A_4} = \sum_{i=1}^{4} t_{G_i} + \sum_{i=1}^{3} o_{G_i}, \quad |O| = o_{G_4}$$

We now prove Part I of Lemma 4.4 by showing that the internal and output probes can be perfectly simulated with $\leq t_{A_4}$ of the input shares $(\mathbf{O}_i)$, and is independent of $|O|$. To simulate the internal probes and output shares of gadgets G_3 and G_4, we require t_{G_3} shares of both inputs of G_3. This is because the t-SNI gadget G_3 stops the propagation of probes at its output (e.g. G_4) to the input shares. Following the flow through gadgets G_2 and G_1, the simulation of G_1 - G_4 of Algorithm 4 requires $|I| = t_{G_1} + t_{G_2} + o_{G_2} + t_{G_3}$ of the input shares $(\mathbf{O}_i)$. Note that without t-SNI refresh G_1, the simulation would require at least $2 \cdot t_{G_3}$ shares of the input and hence would not be sound. As $|I| \leq t_{A_4}$ (no duplicate entries) and independent of o_{G_4}, the first part of Algorithm 4 is t-SNI.

Part II: Gadget G_5 satisfies the t-NI property if the simulator has access to the public value $\mathbf{P}_j^{(3)}$, which is also the output of the full algorithm. As the composition of G_1-G_4 is t-SNI and G_5 is t-NI, its composition and iteration j of the `mCompactKeyGen` algorithm is t-NIo with public output $\mathbf{P}_j^{(3)}$.

Finally, as each iteration j is independent and can be executed in parallel, we can summarize the gadgets in each iteration as a single gadget across all iterations. As a result, the entire Alg. 4 is t-NIo with public output $\left\{\mathbf{P}_j^{(3)}\right\}_{j \in [m]}$. □

4.6 Masked UOV Secret Key Expansion

The secret key expansion in UOV derives the expanded secret key esk, as used during signing, from the compact secret key csk. We propose our masking approach in Algorithm 5, and show a graphical representation in Figure 9. Our strategy consists of using the shared compact secret key to generate the shared expanded key in a masked fashion.

Again, the sensitive part of the secret key csk is the Boolean masked $(seed_{sk,i})_{1 \leq i \leq n}$. Together with the public $seed_{pk}$, they are used to compute the masked expanded secret key components: matrix $(\mathbf{O}_i)$ and matrices $\{(\mathbf{S}_{j,i})_{1 \leq i \leq n}\}_{j \in [m]}$.

Algorithm 5: `mExpandSK`

Data: Boolean shared
compact secret key $csk = (\mathrm{seed}_{\mathrm{pk}}, (\mathrm{seed}_{\mathrm{sk},i})_{1 \leq i \leq n})$
Result: Boolean shared expanded secret key esk

1 $(\mathbf{O}_i) := \mathtt{mExpand}_{\mathsf{sk}}((\mathrm{seed}_{\mathrm{sk},i}))$
2 $\{\mathbf{P}_j^{(1)}, \mathbf{P}_j^{(2)}\}_{j \in [m]} := \mathtt{Expand}_{\mathbf{P}}(\mathrm{seed}_{\mathrm{pk}})$
3 **for** $j = 1$ *upto* m **do**
4 $\quad (\mathbf{S}_{j,i})_{1 \leq i \leq n} := \left(\left(\mathbf{P}_j^{(1)} + \mathbf{P}_j^{(1)T}\right)\mathbf{O}_i\right)$ /* $\mathbf{S}_{j,i} \in \mathbb{F}_q^{l \times m}$ */
5 $\quad \mathbf{S}_{j,1} = \mathbf{S}_{j,1} + \mathbf{P}_j^{(2)}$
6 **return** $esk = ((\mathrm{seed}_{\mathrm{sk},i}), (\mathbf{O}_i), \{\mathbf{P}_j^{(1)}, (\mathbf{S}_{j,i})_{1 \leq i \leq n}\}_{j \in [m]})$

Generation of O and $\{\mathbf{P}_j^{(1)}, \mathbf{P}_j^{(2)}\}_{j \in [m]}$. We refer to Section 4.5, as this procedure (Line 1 - 2) is identical in `mCompactKeyGen`.

Computation of $\left\{\mathbf{S}_j\right\}_{j \in [m]} = \left\{\left(\mathbf{P}_j^{(1)} + \mathbf{P}_j^{(1)T}\right)\mathbf{O} + \mathbf{P}_j^{(2)}\right\}_{j \in [m]}$. The sequence of matrices $\left\{\mathbf{S}_j\right\}_{j \in [m]}$ is computed in a masked fashion, by performing share-wise matrix multiplication and addition. Both $\{\mathbf{P}_j^{(1)}, \mathbf{P}_j^{(2)}\}_{j \in [m]}$ are public values: the sum of $\mathbf{P}_j^{(1)}$ and its transpose is first multiplied with each share of matrix $(\mathbf{O}_i)$ (Line 4). Subsequently, $\mathbf{P}_j^{(2)}$ is added to the first share, to obtain the final m matrices $\{(\mathbf{S}_{j,i})_{1 \leq i \leq n}\}_{j \in [m]}$ (Line 5).

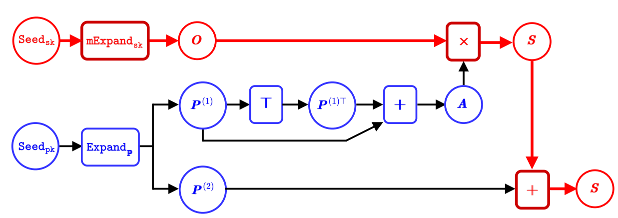

Figure 9: Graphical representation of `mExpandSK()`. Here, red represents masked variables and components, and blue represents unmasked operations and variables. The components introduced or modified due to masking are in dark red color.

4.6.1 *Complexity.*

The run-time and randomness complexity of mExpandSK are:

$$\begin{aligned}T_{\mathsf{mExpandSK}}(l,m,n) &= T_{\mathsf{mExpand_{sk}}}(l,m,n)+T_{\mathsf{Expand}_{\mathbf{P}}}(l,m,n)\\&\quad+m\cdot((l^2n+l^2mn)+(lm))\\&=T_{\mathsf{mExpand_{sk}}}(l,m,n)+T_{\mathsf{mExpand_{pk}}}(l,m,n)\\&\quad+(l^2mn+l^2m^2n+lm^2),\\R_{\mathsf{mExpandSK}}(l,m,n,w)&=R_{\mathsf{mExpand_{sk}}}(l,m,n).\end{aligned}$$

4.6.2 *Security.*

To argue about the first- and high-order security of Algorithm 5, we prove it to be t-NI secure with $n=t+1$ shares, providing resistance against a probing adversary with t probes.

Lemma 4.5. *The gadget* mExpandSK *(Algorithm 5) is t-NI secure.*

Proof. This is a direct result that the operations in a single iteration (multiplication and addition) are linear and performed share-wise (t-NI). If an attacker places t probes across different (independent) iterations, the intermediate values can be simulated with no more number of shares of the input $(\mathbf{O}_i)$. □

4.7 Masked UOV Signature Generation

The UOV signing procedure generates a valid signature σ of an incoming message μ via rejection sampling. As the computation involves the expanded secret key *esk*, we propose to split all secret key and ephemeral components into multiple shares. All computations are performed in a masked manner, as described in Algorithm 6. We also present a graphical version of our masked signature generation operation in Figure 10.

Following its expansion (see previous section), the expanded secret key consists of three Boolean shared components: $(\mathrm{seed}_{\mathrm{sk},i})$, $(\mathbf{O}_i)$ and $\{(\mathbf{S}_{j,i})_{1\le i\le n}\}_{j\in[m]}$. The secret $(\mathrm{seed}_{\mathrm{sk},i})$ is used to derive the vinegar vector $(\mathbf{v}_i)$. In combination with the public matrices $\{\mathbf{P}_j^{(1)}\}_{j\in[m]}$, all components are used to securely compute the unmasked $\mathbf{s}$. Together with a uniformly random string (salt), they form the signature $\sigma=(\mathbf{s},\mathrm{salt})$.

Generation of v. The shares of the secret vinegar vector $\mathbf{v}$ are sampled from a masked PRNG $\mathsf{mExpand}_{\mathbf{v}}$ in Line 4, based on the message μ, the masked secret seed $(\mathrm{seed}_{\mathrm{sk},i})$, a counter and random salt. It is instantiated with a masked shake256(), producing the Boolean shared $(\mathbf{v}_i)$.

Computation of $\mathbf{L}=\mathbf{v}^T\mathbf{S}$. We compute the Boolean shared matrix $(\mathbf{L}_i)$ in a column-wise fashion in Line 5-6. The m Boolean shared matrices $\{(\mathbf{S}_{j,i})_{1\le i\le n}\}_{j\in[m]}$ are multiplied with Boolean shared vector $(\mathbf{v}_i)$, using the SecMatVec gadget. We rely on the transpose property $\mathbf{L}^T=(\mathbf{v}^T\mathbf{S})^T=\mathbf{S}^T\mathbf{v}$.

Computation of $\mathbf{y}=[\mathbf{v}^T\mathbf{P}_j^{(1)}\mathbf{v}]_{j\in[m]}$. We propose to compute the Boolean masked vector $(\mathbf{y}_i)$ using the previously introduced gadget SecQuad (Line 7). Th public matrices $\mathbf{P}_j^{(1)}]_{j\in[m]}$ are first multiplied with the Boolean shared vector $(\mathbf{v}_i)$ and then again with its transpose to obtain $(\mathbf{y}_i)$.

Solving $\mathbf{Lx}=\mathbf{t}-\mathbf{y}$. The system of linear equations is solved using masked Gaussian elimination, using the techniques introduced in [29]. The Boolean shared matrix $(\mathbf{L}_i)$ is first converted to its row-echelon form (SecRowEch, Line 9). Finally, if the resulting (extended) matrix $(\mathbf{T}_i)$ has a non-zero pivot element in each row, the system is back substituted and the public result $\mathbf{x}$ is obtained (SecBackSub, Line 11). We securely unmask and make the output public, as it is a part of the public signature $\mathbf{s}$ (Line 15).

Algorithm 6: mSign

Data: 1. Boolean shared expanded secret key
$esk=((\mathrm{seed}_{\mathrm{sk},i}),(\mathbf{O}_i),\{\mathbf{P}_j^{(1)},(\mathbf{S}_{j,i})_{1\le i\le n}\}_{j\in[m]})$
2. Message μ

Result: Signature σ

1 $\mathrm{salt}\leftarrow\{0,1\}^{\mathrm{salt\_len}}$
2 $\mathbf{t}:=\mathsf{Hash}(\mu\|\mathrm{salt})$
3 **for** $ctr=0$ *upto 255* **do**
4   $(\mathbf{v}_i):=\mathsf{mExpand}_{\mathbf{v}}(\mu\|\mathrm{salt}\|(\mathrm{seed}_{\mathrm{sk},i})\|\mathrm{ctr})$ /* $\mathbf{v}_i\in\mathbb{F}_q^l$ */
5   **for** $j=1$ *upto* m **do** /* $\mathbf{L}^T=(\mathbf{v}^T\mathbf{S})^T=\mathbf{S}^T\mathbf{v}$ */
6     $(\mathbf{L}[:,j]_i)=\mathsf{SecMatVec}((\mathbf{S}_{j,i}^T)_{1\le i\le n},(\mathbf{v}_i))$
7   $(\mathbf{y}_i):=\mathsf{SecQuad}(\{\mathbf{P}_j^{(1)}\}_{j\in[m]},(\mathbf{v}_i))$ /* $\mathbf{y}_i\in\mathbb{F}_q^m$ */
8   $\mathbf{y}_1=\mathbf{y}_1+\mathbf{t}$
9   $(\mathbf{T}_i):=\mathsf{SecRowEch}((\mathbf{L}_i),(\mathbf{y}_i))$ /* $\mathbf{T}_i\in\mathbb{F}_q^{m\times(m+1)}$ */
10   **if** $(T_i)\neq\perp$ **then**
11     $\mathbf{x}:=\mathsf{SecBackSub}((\mathbf{T}_i)$ /* $\mathbf{x}\in\mathbb{F}_q^m$ */
12     $(\mathbf{u}_i):=(\mathbf{v}_i+\mathbf{O}_i\mathbf{x})$
13     $\mathbf{w}:=\mathsf{FullAdd}((\mathbf{u}_i))$ /* $\mathbf{w}\in\mathbb{F}_q^l$ */
14     $\mathbf{s}:=\begin{bmatrix}\mathbf{w}\\\mathbf{x}\end{bmatrix}$
15   **return** $\sigma=(\mathbf{s},\mathrm{salt})$
16 **return** $\perp$

Computation and unmasking of w. The second part of the signature, $\mathbf{w}$, is computed in a share-wise fashion: each share of $(\mathbf{v}_i)$ is added to the product of the public vector $\mathbf{x}$ and Boolean shared matrix $(\mathbf{O}_i)$ in Line 12. Finally, the resulting shares are securely combined (FullAdd, Line 13) and the vector $\mathbf{w}$ is made public as part of the signature $\mathbf{s}$.

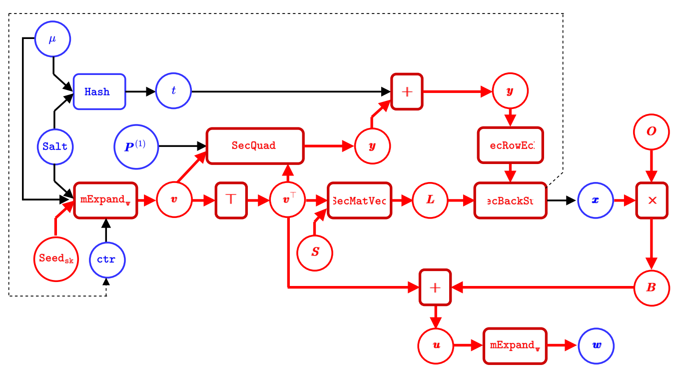

Figure 10: Graphical representation of mSign(). Here, red represents masked variables and components, and blue represents unmasked operations and variables. The components introduced or modified due to masking are in dark red color.

4.7.1 *Complexity.*

The full randomness complexity computation is included in Appendix C.

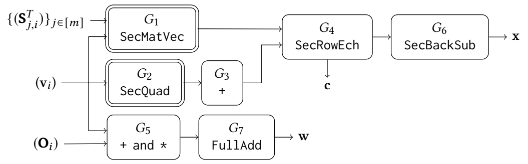

Figure 11: An abstract diagram of an iteration ctr in `mSign` (Alg. 6). The t-NI gadgets are depicted with a single border, the t-SNI gadgets with a double border.

4.7.2 Security. We now discuss the first- and high-order security of Algorithm 6 and prove it to be t-NIo secure with $n = t+1$ shares and public outputs **s** and **c**. The signature **s** is public, while **c** is made public by gadget `SecRowEch` and indicates if all pivot-elements are non-zero. As a result, our masked algorithm provides resistance against a probing adversary with t probes.

LEMMA 4.6. *The gadget* `mSign` *(Algorithm 6) is t-NIo secure with public outputs* **s** *(***w***,***x***) and* **c**.

Proof. We model a single iteration of Algorithm 6 as a composition of t-(S)NI gadgets, which is visually shown in Figure 11. Apart from the gadgets listed in Table 2, we model the share-wise operations in Line 8 and 12 as t-NI gadgets G_3 and G_5, respectively. It is trivial to show that linear operations are t-NI. We also model all iterations in the inner loop (Line 5-6) as a single t-SNI gadget G_1. As the iterations are independent and we define probes on a column level, simulation is successful. Each iteration produces one of m independent columns of $(\mathbf{T}_i)$ and is assumed to be executed in parallel. We note that the algorithm and its security proof are independent of the specific masked implementation used for the PRNG $\mathtt{mExpand}_{\mathbf{v}}$. An adversary can probe any intermediate values in any gadget (t_{G_i}) and their output shares o_{G_i}. The total number of probes in Algorithm 6 is

$$t_{A_6} = \sum_{i=1}^{7} t_{G_i} + \sum_{i=1}^{5} o_{G_i}$$

We now show that all probes in a single iteration of `mSign` can be simulated with no more number of shares of its inputs ($|I|$): $|I| \leq t_{A_6}$ if the simulator has access to **x**,**w** and **c**. The simulation of t_{G_6} and t_{G_7} intermediate probes requires an equal amount of shares of the outputs of G_4 and G_5, respectively. This is due to the t-NI property of both gadgets. Similarly, the simulation of $t_{G_4}+o_{G_4}$ probes requires $t_{G_4}+o_{G_4}$ shares of both the output of G_3 and G_1, and giving the simulator access to **c**. The simulation of $t_{G_5}+o_{G_5}$ probes requires the same amount of shares of inputs $(\mathbf{v}_i)$ and $(\mathbf{O}_i)$. Due the t-SNI property of G_1 and G_2, the simulation of probed intermediate values and output shares only requires t_{G_1} and t_{G_2} shares of inputs $\{(\mathbf{S}_{j,i}^T)\}_{j\in[m]}$ and/or $(\mathbf{v}_i)$, respectively. We now follow the flow from the output to the input and sum all required shares of the input for simulation of Algorithm 6: $|I| = t_{G_1} + t_{G_2} + t_{G_5} + o_{G_5} + t_{G_7} \leq t_{A_6}$. As a result, the iteration is t-NI secure with public outputs **s** and **c**.

Finally, we note that the signing procedure only requires multiple iterations if the system of linear equations is unsolvable and no unique solution **x** can be found. In that case, all masked computations are performed again using a new vinegar vector $(\mathbf{v}_i)$ and thus are different from the previous iteration. As different iterations are independent, the entire outer loop (Line 3-15) is also t-NI secure with public outputs **s** and **c**. □

5 IMPLEMENTATION RESULTS

This section presents the implementation results of masked UOV algorithms for first-order. We implemented our masked key generation and signature generation algorithm for ARM Cortex-M4 using the popular PQM4 [22, 23] framework. We have used the NUCLEO-L4R5ZI board and arm-none-eabi-gcc compiler with version 10.3.1. We have used on-chip TRNG to generate random bytes. Therefore, our performance results include random bytes generation, too.

Due to the memory constraints of the NUCLEO-L4R5ZI board (640 KB RAM), we are restricted to NIST security level 1 parameters, i.e., UOV-Ip. To run our masked implementations along with benchmarking code, we choose the pkc variant of UOV-Ip. In the pkc variant, `CompactKenGen` and `ExpandSk` are combined in key generation. We have adopted the masked implementation of `shake256` from [6] for the PRNG in $\mathtt{mExpand}_{sk}$ and $\mathtt{mExpand}_{\boldsymbol{v}}$.

We present the performance results and memory consumption of our masked implementation of UOV-Ip with the unmasked implementation of UOV-Ip [23] in Table 3. Here, we report averages of 10 measurements of all the algorithms. Our first-order masked key generation and signature generation implementations use 1,055,464 and 25,587 (1000×)cycles, introducing 2.7× and 3.5× overhead over the unmasked implementation, respectively. Additionally, our masked implementation utilizes 5,796 and 10,160 bytes of stack memory for the key generation and signature generation algorithm, introducing 1.3× and 1.9× overhead over the unmasked implementation.

6 CONCLUSIONS

The multivariate digital signature UOV is susceptible to side-channel attacks. We thoroughly analyze the sensitivity of all variables and functions, which could lead to such physical attacks. As a countermeasure, we propose arbitrary-order masked key generation and signature generation algorithms of UOV. Finally, we implemented our first-order masking algorithms for UOV-Ip on the ARM Cortex-M4 microcontroller. Our first-order masked UOV implementations have 2.7× and 3.6× performance overhead compared to the unmasked scheme for key generation and signature generation algorithms. Finally, we would like to note that our approach is not limited to the UOV scheme but can be extended to other UOV-based multivariate schemes, such as Mayo, QR-UOV, SNOVA, and MQ-Sign.

Acknowledgements. This work was partially supported by Horizon 2020 ERC Advanced Grant (101020005 Belfort), Horizon Europe (101070008 ORSHIN), CyberSecurity Research Flanders with reference number VOEWICS02, BE QCI: Belgian-QCI (3E230370) (see beqci.eu), and Intel Corporation. A.G. thanks to TCS research fellowship.

REFERENCES

[1] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. 2022. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. Online. Accessed 26th January, 2024. https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413-upd1.pdf

[2] Thomas Aulbach, Fabio Campos, Juliane Krämer, Simona Samardjiska, and Marc Stöttinger. 2023. Separating Oil and Vinegar with a Single Trace Side-Channel

Table 3: Comparing the performance and memory consumption of masked implementations of UOV-Ip with the unmasked ones

Scheme: UOV-Ip	Key-generation			Sign		
	unmasked	masked		unmasked	masked	
Speed (1000× cycles)	390,347	**1,055,464**	**(2.7×)**	7,127	**25,587**	**(3.6×)**
Memory (bytes)	4,484	**5,796**	**(1.3×)**	5,232	**10,160**	**(1.9×)**

Assisted Kipnis-Shamir Attack on UOV. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2023, 3 (2023), 221–245. https://doi.org/10.46586/TCHES.V2023.I3.221-245

[3] Thomas Aulbach, Fabio Campos, and Juliane Krämer. 2024. SoK: On the Physical Security of UOV-based Signature Schemes. Cryptology ePrint Archive, Paper 2024/1818. https://eprint.iacr.org/2024/1818

[4] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. 2016. Strong Non-Interference and Type-Directed Higher-Order Masking. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 116–129. https://doi.org/10.1145/2976749.2978427

[5] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. 2018. Masking the GLP lattice-based signature scheme at any order. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37.* Springer, 354–384.

[6] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. 2020. A Side-Channel Resistant Implementation of SABER. Cryptology ePrint Archive, Paper 2020/733. https://doi.org/10.1145/3429983

[7] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. 2019. The SPHINCS$^+$ Signature Framework. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 2129–2146. https://doi.org/10.1145/3319535.3363229

[8] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. 2023. MAYO Specification Document. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/mayo-spec-web.pdf

[9] 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. 2023. UOV: Unbalanced Oil and Vinegar Algorithm Specifications and Supporting Documentation Version 1.0. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/UOV-spec-web.pdf

[10] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology — CRYPTO' 99*, Michael Wiener (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 398–412.

[11] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. 2014. Secure conversion between boolean and arithmetic masking of any order. In *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer, 188–205.

[12] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. 2022. High-order Polynomial Comparison and Masking Lattice-based Encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (11 2022), 153–192. https://doi.org/10.46586/tches.v2023.i1.153-192

[13] Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. 2023. Improved Gadgets for the High-Order Masking of Dilithium. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2023, 4 (Aug. 2023), 110––145. https://doi.org/10.46586/tches.v2023.i4.110-145

[14] Jean-Sébastien Coron and Lorenzo Spignoli. 2021. Secure Wire Shuffling in the Probing Model. In *Advances in Cryptology – CRYPTO 2021*, Tal Malkin and Chris Peikert (Eds.). Springer International Publishing, Cham, 215–244.

[15] Leo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. 2017. CRYSTALS – Dilithium: Digital Signatures from Module Lattices. Cryptology ePrint Archive, Paper 2017/633. https://eprint.iacr.org/2017/633 https://eprint.iacr.org/2017/633.

[16] Sebastian Faust, Vincent Grosso, Santos Pozo, Clara Paglialonga, and François-Xavier Standaert. 2018. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (08 2018), 89–120. https://doi.org/10.46586/tches.v2018.i3.89-120

[17] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. 2020. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. https://falcon-sign.info/falcon.pdf.

[18] Hiroki Furue, Yasuhiko Ikematsu, Fumitaka Hoshino, Tsuyoshi Takagi, Kan Yasuda, Toshiyuki Miyazawa, Tsunekazu Saito, and Akira Nagai. 2023. QR-UOV Specification Document. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/qruov-spec-web.pdf

[19] Hannes Gross, Rinat Iusupov, and Roderick Bloem. 2018. Generic Low-Latency Masking in Hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (05 2018), 1–21. https://doi.org/10.46586/tches.v2018.i2.1-21

[20] Hannes Gross, Stefan Mangard, and Thomas Korak. 2016. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security* (Vienna, Austria) *(TIS '16)*. Association for Computing Machinery, New York, NY, USA, 3. https://doi.org/10.1145/2996366.2996426

[21] Yuval Ishai, Amit Sahai, and David Wagner. 2003. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23.* Springer, 463–481.

[22] Matthias J. Kannwischer, Markus Krausz, Richard Petri, and Shang-Yi Yang. 2024. pqm4: Benchmarking NIST Additional Post-Quantum Signature Schemes on Microcontrollers. Cryptology ePrint Archive, Paper 2024/112. https://eprint.iacr.org/2024/112

[23] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. [n. d.]. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[24] Aviad Kipnis, Jacques Patarin, and Louis Goubin. 1999. Unbalanced Oil and Vinegar Signature Schemes. In *Advances in Cryptology — EUROCRYPT '99*, Jacques Stern (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 206–222.

[25] NIST. 2023. NIST Announces Additional Digital Signature Candidates for the PQC Standardization Process. Online. Accessed 10th November, 2024. https://csrc.nist.gov/news/2023/additional-pqc-digital-signature-candidates

[26] NIST. 2024. FIPS 204 Module-Lattice-Based Digital Signature Standard. Online. Accessed 10th November, 2024. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf

[27] NIST. 2024. FIPS 205 Stateless Hash-Based Digital Signature Standard. Online. Accessed 10th November, 2024. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf

[28] NIST. 2024. PQC digital signature second round announcement. Online. Accessed 10th November, 2024. https://csrc.nist.gov/News/2024/pqc-digital-signature-second-round-announcement

[29] Quinten Norga, Suparna Kundu, Uttam Kumar Ojha, Anindya Ganguly, Angshuman Karmakar, and Ingrid Verbauwhede. 2024. Masking Gaussian Elimination at Arbitrary Order, with Application to Multivariate- and Code-Based PQC. Cryptology ePrint Archive, Paper 2024/1777. https://eprint.iacr.org/2024/1777

[30] Aesun Park, Kyung-Ah Shim, Namhun Koo, and Dong-Guk Han. 2018. Side-channel attacks on post-quantum signature schemes based on multivariate quadratic equations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (Aug. 2018), 500–523. https://doi.org/10.13154/tches.v2018.i3.500-523

[31] Pierre Pébereau. 2023. One vector to rule them all: Key recovery from one vector in UOV schemes. Cryptology ePrint Archive, Paper 2023/1131. https://eprint.iacr.org/2023/1131

[32] Quantum-Resistant Cryptography Research Group. 2024. KpqC Competition Round 2. https://www.kpqc.or.kr/competition_02.html Accessed: 2024-10-30.

[33] Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. 2019. Efficiently Masking Binomial Sampling at Arbitrary Orders for Lattice-Based Crypto. In *Public-Key Cryptography – PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II* (Beijing, China). Springer-Verlag, Berlin, Heidelberg, 534–564. https://doi.org/10.1007/978-3-030-17259-6{_}18

[34] Kyung-Ah Shim, Jeongsu Kim, and Youngjoo An. 2023. MQ-Sign: A New Post-Quantum Signature Scheme based on Multivariate Quadratic Equations: Shorter and Faster. https://www.kpqc.or.kr/images/pdf/MQ-Sign.pdf

[35] Lih-Chung Wang, Chun-Yen Chou, Jintai Ding, Yen-Liang Kuan, Ming-Siou Li, Bo-Shu Tseng, Po-En Tseng, and Chia-Chun Wang. 2023. SNOVA Specification Document. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/SNOVA-spec-web.pdf

[36] Haibo Yi and Zhe Nie. 2018. Side-channel security analysis of UOV signature for cloud-based Internet of Things. *Future Gener. Comput. Syst.* 86 (2018), 704–708. https://doi.org/10.1016/J.FUTURE.2018.04.083

APPENDICES

A Proof Lemma 4.1

The security proof follows from the potential observations that a probing adversary can make. We note that probes are defined on the *coefficient*-level: the output of the gadget is a coefficient and the inputs are vectors, consisting of l *independent* coefficients. We now show that all potential observations can be perfectly simulated using a limited amount of shares of each of the l (independent) inputs.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be a set of t observations made by an adversary on the internal and output values, respectively, where $|\mathcal{I}| = t_{A_1}$, such that $t_{A_1} + |\mathcal{O}| \leq t$. We construct a perfect simulator of the adversary's probes, which makes use of at most t_{A_1} shares of the secret input coefficients $\mathbf{x}[k]$ and $\mathbf{y}[k]$ $(1 \leq k \leq l)$.

Let $w_1,\ldots,w_t$ be the set of probed values. We classify the internal wires in the following groups:

(1) $x[k]_i, y[k]_j, x[k]_i y[k]_j$ at iteration i,j,k,
(2) $u_{ij}, u_{ij} + x[k]_i y[k]_j$ at iteration i,j,k,
(3) $x[k]_j, y[k]_i, x[k]_j y[k]_i$ at iteration i,j,k,
(4) $u_{ji}, u_{ji} + x[k]_j y[k]_i$ at iteration i,j,k,
(5) $x[k]_i, y[k]_i, x[k]_i y[k]_i$ at iteration i,k,
(6) $w_i, w_i + x[k]_i y[k]_i$ at iteration i,k,
(7) $u_{ij} + r_{ij}$ with $i,j = 1,\ldots,t+1$,

The output variables are the final values of (z_i).

We define two arrays of sets of indices I_k and J_k $(1 \leq k \leq l)$ such that $|I_k| \leq t_{A_1}$ and $|J_k| \leq t_{A_1}$ and the values of the probes can be perfectly simulated given only knowledge of $(\mathbf{x}[k]_i)_{i \in I_k}$ and $(\mathbf{y}[k]_i)_{i \in J_k}$. The sets are constructed as follows.

- Initially all I_k and J_k are empty $(1 \leq k \leq l)$.
- For every probe as in group (1) add i to I_k and j to J_k.
- For every probe as in group (2) and (7) add i to I_m and j to J_m with $m = 1,\ldots,k$.
- For every probe as in group (3) add j to I_k and i to J_k.
- For every probe as in group (4) add j to I_m and i to J_m with $m = 1,\ldots,k$.
- For every probe as in group (5) add i to I_k and J_k.
- For every probe as in group (6) add i to I_m and J_m with $m = 1,\ldots,k$.

An adversary is allowed to make t_{A_1} internal probes at most, thus it holds that $|I_k| \leq t_{A_1}$ and $|J_k| \leq t_{A_1}$ $(1 \leq k \leq l)$.

We now construct the simulator with the probed wires denoted w_h with $h = 1,\ldots,t$ and show it is able to simulate any internal wire w_h. For each variable r_{ij} entering in the computation of any probe, the simulator assigns a random value.

1. For each observation as in group (1) (or (3)), by definition of I_k and J_k the simulator has access to $x[k]_i$ and $y[k]_j$ (or $x[k]_j$ and $y[k]_i$, respectively) and thus the values are perfectly simulated.

2. For each observation as in group (2) (or (4)), by definition of $\{I_m\}_{1 \leq m \leq k}$ and $\{J_m\}_{1 \leq m \leq k}$ the simulator has access to $x[m]_i$ and $y[m]_j$ (or $x[m]_j$ and $y[m]_i$, respectively) for $m = 1,\ldots,k$ and thus the values are perfectly simulated.

3. For each observation as in group (5), by definition of I_k and J_k the simulator has access to $x[k]_i$ and $y[k]_i$ and thus the values are perfectly simulated.

4. For each observation as in group (6), by definition of $\{I_m\}_{1 \leq m \leq k}$ and $\{J_m\}_{1 \leq m \leq k}$ the simulator has access to $x[m]_i$ and $y[m]_i$ for $m = 1,\ldots,k$ and thus the values are perfectly simulated.

5. For each observation as in group (7), by definition of $\{I_k\}_{1 \leq k \leq l}$ and $\{J_k\}_{1 \leq k \leq l}$ the simulator has access to $x[k]_i$ and $y[k]_j$ for $k = 1,\ldots,l$ and we distinguish three cases:

- If $i = j$, the simulator assigns r_{ii} to 0 and perfectly simulates the value w_h using $x[k]_i$ and $y[k]_i$ for $k = 1,\ldots,l$.
- If $j \in I$ and $i \in J$, then by definition the adversary has also probed u_{ji} and thus a value containing in its computation the random value r_{ij}. The simulator then perfectly simulates w_h using $x[k]_i$ and $y[k]_j$ for $k = 1,\ldots,l$ and the r_{ij} assigned previously.
- In all other cases, r_{ij} does not enter in the computation of any other probe and w_h is assigned a fresh random value and thus perfectly simulated.

We now consider the observations of the output values. We distinguish two cases:

- If an intermediate sum is also observed, then the previously probed partial sums are already simulated. By definition of the gadget, there always exists one random bit r_{op} in w_h which does not appear in the computation of any other observed element. Thus, the simulator can assign a fresh random value to w_h.
- If no internal values have been probed by an adversary, then by definition of the gadget, each output share contains t random values and at most one of them can enter in the computation of each other output variable z_i. An adversary may have probed $t-1$ other values and thus there exists one random value r_{op} in w_h which does not enter in the computation of any other observed value. The simulator can thus simulate w_h using a fresh random value, completing the proof. ◻

B Auxiliary Algorithms

Algorithm 7: `FullAdd`, from [5, 11]

Data: A Boolean sharing (y_i)

Result: Unmasked value y such that $y = \sum_{i=1}^{n} y_i$

```
(a_i) := SecREF((y_i))          /* free-t-SNI */
y := a_1 + ··· + a_n
return y
```

Algorithm 8: SecRowEch, from [29]

Data: 1. A Boolean sharing $(\mathbf{A}_i)$ of matrix $\mathbf{A} \in \mathbb{F}_q^{m \times m}$
2. A Boolean sharing $(\mathbf{b}_i)$ of the vector $\mathbf{b} \in \mathbb{F}_q^{m}$
Result: Masked conversion to row echelon form or $\perp$

$(\mathbf{T}_i) := [\mathbf{A}_i \mid \mathbf{b}_i]$ /* $\mathbf{T}_i \in \mathbb{F}_q^{m \times (m+1)}$ */
for $j = 1$ *upto* m **do**
  ## Try to make pivot $(\mathbf{T}[j,j])$ non-zero
  for $k = j+1$ *upto* m **do**
    $(z_i) :=$ SecNonzero$((\mathbf{T}[j,j]_i))$
    $(z_i) =$ SecNOT$((z_i))$
    $(\mathbf{T}[j,j{:}m+1]_i) =$ SecCondAdd$((\mathbf{T}[j,j{:}m+1]_i), (\mathbf{T}[k,j{:}m+1]_i), (z_i))$
  ## Check if pivot is non-zero
  $(t_i) :=$ SecNonzero$((\mathbf{T}[j,j]_i))$
  $\mathbf{c}[j] :=$ FullAdd$((t_i))$
  if $\mathbf{c}[j] \neq 0$ **then**
    ## Multiply row j with the inverse of its pivot
    $(p_i) :=$ B2Minv$((\mathbf{T}[j,j]_i))$
    $(\mathbf{T}[j,j{:}m+1]_i) =$ SecScalarMult$((\mathbf{T}[j,j{:}m+1]_i), (p_i))$
    ## Subtract scalar multiple of row j from the rows below
    for $k = j+1$ *upto* m **do**
      $(s_i) :=$ SecREF$((\mathbf{T}[k,j]_i))$
      $(\mathbf{T}[k,j{:}m+1]_i) =$ SecMultSub$((\mathbf{T}[j,j{:}m+1]_i), (\mathbf{T}[k,j{:}m+1]_i), (s_i))$
  else return $\perp$
return $(\mathbf{T}_i)$

Algorithm 9: SecBackSub, from [29]

Data: A Boolean sharing $(\mathbf{T}_i) = [\mathbf{A}_i | \mathbf{b}_i]$ of matrix $\mathbf{A} \in \mathbb{F}_q^{m \times m}$ and vector $\mathbf{b} \in \mathbb{F}_q^{m}$.
Result: Unique, public solution $\mathbf{x} \in \mathbb{F}_q^{m}$ such that $\mathbf{A}\mathbf{x} = \mathbf{b}$

for $j = m$ *downto* 2 **do**
  $\mathbf{x}[j] =$ FullAdd$((\mathbf{b}[j]_i))$
  for $k = 1$ *upto* $j-1$ **do**
    $(\mathbf{b}[k]_i) := (\mathbf{b}[k]_i + \mathbf{x}[j] \cdot \mathbf{A}[k,j]_i)$
$\mathbf{x}[1] =$ FullAdd$((\mathbf{b}[1]_i))$
return $\mathbf{x}$

C Complexity Algorithm 6 (mSign())

The run-time and randomness complexity of mSign are:

$$
\begin{aligned}
T_{\mathsf{mSign}}(l,m,n) = {} & 1 + T_{\mathsf{Hash}}(m,n) + \mathtt{ctr} \cdot (T_{\mathsf{mExpand}_{\boldsymbol{v}}}(l,n) \\
& + (m \cdot T_{\mathsf{SecMatVec}}(l,m,n)) + (T_{\mathsf{SecQuad}}(l,m,n)) \\
& + (m) + (T_{\mathsf{SecRowEch}}(m,n)) + (T_{\mathsf{SecBackSub}}(m,n)) \\
& + (lmn + ln) + (l \cdot T_{\mathsf{FullAdd}}(n))) \\
= {} & 1 + T_{\mathsf{Hash}}(m,n) + \mathtt{ctr} \cdot (T_{\mathsf{mExpand}_{\boldsymbol{v}}}(l,n) \\
& + (lm^2n^3 - lm^2n^2 + lm^2n + \frac{3}{2}m^2n^3 - \frac{1}{2}m^2n^2 \\
& - m^2n) + (\frac{3}{2}ln^2 - \frac{3}{2}ln) + (\frac{1}{2}l^2m^2n + \frac{1}{2}l^2mn) \\
& + (lmn^3 - lmn^2 + lmn + \frac{3}{2}mn^3 - \frac{1}{2}mn^2 - mn) \\
& + (m) + (\frac{m^2-m}{2} \cdot (((5n^2 + 2n - 1) + \lceil \log(w+1) \rceil \cdot \\
& (5n^2 - n + 2)) + 1) + \frac{2m^3 + 3m^2 + m}{6} \cdot (5n^2 - 3n) + m \cdot \\
& ((5n^2 + 2n - 1) + \lceil \log(w+1) \rceil \cdot (5n^2 - n + 2)) \\
& + m \cdot \frac{3n^2 - n - 2}{2} + m + m \cdot \frac{5n^2 - 5n + 4}{2} \\
& + \frac{m^2 + 3m}{2} \cdot (5n^2 - 3n) + \frac{m^2 - m}{2} \cdot \\
& \frac{3n^2 - 3n}{2} + \frac{2m^3 + 3m^2 + m}{6} \cdot \frac{7n^2 - 3n}{2}) \\
& + (\frac{3}{2}n^2m - \frac{3}{2}mn - m + m^2n) + (lmn + ln) \\
& + ((\frac{3}{2}ln^2 - \frac{3}{2}ln + ln - l)),
\end{aligned}
$$

$$
\begin{aligned}
R_{\mathsf{mSign}}(l,m,n,w) = {} & \mathtt{ctr} \cdot (R_{\mathsf{mExpand}_{\boldsymbol{v}}}(l,n,w) + (m \cdot R_{\mathsf{SecMatVec}}(l,m,n,w)) \\
& + (R_{\mathsf{SecQuad}}(l,m,n,w)) + (R_{\mathsf{SecRowEch}}(m,n,w)) \\
& + (R_{\mathsf{SecBackSub}}(m,n,w)) + (l \cdot R_{\mathsf{FullAdd}}(n,w))) \\
= {} & \mathtt{ctr} \cdot (R_{\mathsf{mExpand}_{\boldsymbol{v}}}(l,n,w) + (\frac{1}{2}m^2n^3w - \frac{1}{2}m^2n^2w) \\
& + (\frac{1}{2}ln^2w + \frac{1}{2}lnw) + (\frac{1}{2}mn^3w - \frac{1}{2}mn^2w) \\
& + (\frac{m^2-m}{2} \cdot \frac{\lceil \log(w+1) \rceil^2 - \lceil \log(w+1) \rceil}{2} \cdot \\
& (n^2 - n) + \frac{2m^3 + 3m^2 + m}{6} \cdot (n^2 - n)w + m \cdot \\
& \frac{\lceil \log(w+1) \rceil^2 - \lceil \log(w+1) \rceil}{2} \cdot (n^2 - n) \\
& + m \cdot \frac{(n^2 - n)w}{2} + m \cdot \frac{n^2 - n}{2} \cdot w + \frac{m^2 + 3m}{2} \cdot \\
& (n^2 - n)w + \frac{m^2 - m}{2} \cdot (\frac{n^2 - n}{2} \cdot w) + \frac{2m^3 + 3m^2 + m}{6} \cdot \\
& \frac{n^2 - n}{2} \cdot w) + (\frac{(n^2 - n)mw}{2}) + (\frac{1}{2}ln^2w - \frac{1}{2}lnw)).
\end{aligned}
$$