mUOV: Masking the Unbalanced Oil and Vinegar Digital 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

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

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

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

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

ABSTRACT

In the recent search for additional post-quantum designs, multivariate quadratic equations (MQE) based designs have been receiving attention due to their small signature sizes. Unbalanced Oil and Vinegar (UOV) is an MQE-based digital signature (DS) scheme proposed over two decades ago. Although the mathematical security of UOV has been thoroughly analyzed, several practical side-channel attacks (SCA) have been shown on UOV based DS schemes. In this work, we perform a thorough analysis to identify the variables in UOV based DS schemes that can be exploited with passive SCA, specifically differential power attacks (DPA). Secondly, we introduce masking as a countermeasure to protect the sensitive components of UOV based schemes. We propose efficient masked gadgets for all the critical operations, including the masked dot-product and matrix-vector multiplication. We show that our gadgets are secure in the t-probing model through formal proofs, mechanically verified using the `maskVerif` tool. We implemented and demonstrated the practical feasibility of our arbitrary-order masking algorithms for UOV-Ip and UOV-III. We show that the masked signature generation of UOV-Ip performs up to 62% better than Dilithium2 or ML-DSA and 99% better than Falcon 512 or FN-DSA. In addition, the security of our implementation is practically validated using the test vector leakage assessment (TVLA) methodology.

CCS CONCEPTS

• **Security and privacy** → **Digital signatures**; **Side-channel analysis and countermeasures**.

KEYWORDS

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

1 INTRODUCTION

The National Institute of Standards and Technology (NIST) recently published the first set of post-quantum (PQ) digital signature algorithm (DSA) standards [1]. Currently, this set constitutes ML-DSA [33] (CRYSTALS-Dilithium) and SLH-DSA [34] (Sphincs+), while FN-DSA [23] (Falcon) will be released in the near future. One common characteristic of these standard schemes is their large signature size, which is multiple orders ($10-120\times$) the size of existing elliptic curve discrete signature algorithm (EC-DSA) signatures. This creates serious bottlenecks for many critical applications. For example, in a chain-of-trust-based authentication in the transport layers security (TLS) where an entity, e.g a website, is authenticated by a series of certificates i.e. *root certificate-intermediate certificate^l-leaf certificate*, $l \geq 1$, this results in a huge blowup in the required transmission bandwidth. This problem has led to the proposal of some unorthodox approaches, such as KEMTLS [44] or KEMTLS with redistributed public-keys [45].

Nonetheless, the adoption and integration of current PQC standards pose a significant challenge for devices with constrained resources, such as the Internet of Things devices, sensor nodes, etc. These devices use constrained radio networks (CRN) such as Low-Power Personal Area Networks (LPPANs) g.s Bluetooth Low Energy with a range from a few centimeters to a few meters, and Low Power Wide Area Networks (LPWANs) such as LoRaWAN, Sigfox, IEEE 802.11ah, etc. which have ranges of several kilometers. Due to operational constraints, CRNs have very small frame sizes, ultra-low speeds, and very high latency. For example, LoRaWAN has 51 bytes frames and 11 bytes frames in Europe and the United States, respectively. The small frame size combined with approximately 1% duty cycle i.e. a device sends data for 36 seconds and waits for an hour, basically means that it may take a few days (or even more in case of transmission errors) to transmit the 2.4KB signature payload of ML-DSA. Furthermore, in resource-constrained devices (RCD), the energy cost for radio transmission is significantly larger compared to the computational costs [20, 31, 38]. Similarly, the signature size of SLH-DSA and its large signing time, makes the scheme unsuitable for integration into RCDs. Interestingly, FN-DSA produces relatively small signatures (666 bytes) but due its complex data structures (Falcon tree) and floating point arithmetic it is challenging to implement (securely) on embedded devices [1].

Table 1: Signature sizes (bytes) of pre- and post-quantum DSAs.

Algorithm	EC-DSA	ML-DSA	FN-DSA	SLH-DSA	UOV
Assumption	EC DLP	SIS	NTRU	Hash	MQE
Signature [B]	64	2420	666	7856	**96**

As a result, NIST explicitly mentions the need for PQ DSAs suitable for RCDs in their call for the standardization of additional PQ DSAs [32], which has advanced to its second round. Signature schemes based on the hardness of solving multivariate quadratic

*Two authors contributed equally to this research.

equations (MQE) [9, 13, 40, 41], which is an NP-complete problem [29] offer relatively much smaller signature sizes compared to the PQ DS schemes based on other hard problems and only slightly larger than EC-DSA (Table 1). In this paper, we focus on the Unbalanced Oil and Vinegar (UOV) DSA [11], which was originally proposed by Kipnis et al. [30] and is a Round 2 candidate. Several other MQE-based DSAs have been constructed using the UOV framework. Among them, QR-UOV [24], SNOVA [49], and MAYO [10], which have also advanced to Round 2 of the NIST additional DSA standardization process [36]. Another scheme, MQ-Sign [47], was a finalist candidate in the recently concluded Korean PQC standardization procedure [42].

Side-Channel attacks (SCA) exploit the physical phenomena of devices which are performing cryptographic operations dependent on secret key material. Examples of such phenomena include secret-dependent execution time, power consumption, electromagnetic radiation, etc. For a real-world deployment of cryptographic schemes, especially those running on small RCD, such attacks are one of the most potent threats [3, 39, 51]. Therefore, integrating countermeasures against side-channel attacks is a crucial and necessary step for a real world deployment of any cryptographic scheme. NIST also stressed on this criterion in its call for standardization [32]. Masking is a well-known and provably secure countermeasure against differential power attacks (DPA), first introduced by Chari et al. [12]. Here, secret values are split into multiple randomized shares and computations are performed in such a way that an adversary who is not able to recover all shares, cannot construct the full secret.

Contribution. In this work, we present a complete analysis and methodology for masking and protecting the full UOV digital signature scheme against first- and higher-order DPAs, which we formally and empirically validate. There are many works that have proposed secure masking algorithms for current NIST PQC DSA standards [8, 14, 18]. To the best of our knowledge and in spite of being in existence for a long time, there are no masking schemes for the UOV DSA. Therefore, the primary motivation for this work is to close this gap in research. More specifically, our contributions are below.

- First, we perform a systematic and rigorous sensitivity analysis on the complete UOV scheme. We identify critical variables and operations that require protection against DPAs.
- Second, we propose novel masked gadgets for all sub-operations and provide formal proofs in the t-probing model, which are verified using the `maskVerif` tool [5]. We propose an efficient, arbitrary-order masked algorithm for matrix-vector multiplication based on our `SecDotProd` gadget. We propose a *lazy compression* technique, which requires only a single, costly mask refresh when performing the masked dot product between two vectors of l coefficients, compared to the standard approach requiring l refresh operations. Our approach allows the delay of the expensive share re-masking and final compression and performs it once, combining all cross-products at once. Our gadget allows us to construct efficient and secure matrix-vector multiplications, on which all MQE-based schemes heavily rely.
- Third, we combine our novel and efficient gadgets with methods from prior work to present an open-sourced, arbitrary-order masked implementation of all sensitive UOV routines of key generation, secret key expansion, and signature generation.
- Fourth, we experimentally validate the security of the first-order implementations of our proposed gadgets using test vector leakage assessment (TVLA) methodology (1M executions). We demonstrate how to eliminate physical leakages due to micro-architectural effects in masked implementations and identify compiler optimization flags, which allow for the use of aggressive compiler optimization (`-o3`) without impacting security.
- Fifth, we compare the performances of masked UOV implementations with other PQ DSAs. Additionally, we demonstrate the benefits of using our techniques and implementation to deploy PQ-secure cryptography in embedded environments.

We make the source code of our implementation and the scripts for formal verification of our proofs in the t-probing model (`maskVerif`) available for reviewers at https://anonymous.4open.science/r/mUOV-CCS-EB53/. (see Appendix A).

Outline. We first introduce the notation and definitions used throughout this work and a description of the UOV DSA in Section 2. Subsequently, in Section 3, we analyze the UOV scheme from side-channel perspective and analyze which components are sensitive and require protection against DPAs. We propose novel, arbitrary-order masked gadgets for efficient vector & matrix arithmetic in Section 4. In Section 5, we propose `mUOV`, which includes the masked key generation and signature generation algorithms. We present and discuss our implementation, including an extensive performance and security evaluation in Section 6. Finally, we conclude with applications of masked UOV in Section 7.

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. In this work, we use Boolean masking, where $x = x_1 + \ldots + x_n$, and the addition is a logical XOR ($\oplus$).

2.2 Masking

Ishai et al. [28] 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. Barthe et al. [6] introduced several security notions, which allow us to prove the probing security of the composition of sub-operations (*gadgets*). We now recall the security notions used in this work, as presented in [43].

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 $t_1+t_2 \leq t$ 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.

Definition 2.2 (free-t-Strong-Non-Interference (free-t-SNI) security [19]). 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 $t_1 \leq t$ there exists a subset I of input indices with $|I| \leq t_1$, 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 [7]). A gadget with public output b and m_i input sharings is t-NIo secure if, for any set of $t_1 \leq t$ intermediate variables, there exists a subset I of input indices with $|I| \leq t_1$, such that t_1 intermediate variables can be perfectly simulated from $x_{|I}$ and b.

2.3 UOV-DSA

This work specifically targets the UOV digital signature scheme, as submitted to the latest NIST standardization process [11]. Its Round 2 specification defines three variants: `classic`, `pkc`, and `pkc+skc`. These variants are designed to offer different trade-offs between memory utilization and performance. The `classic` variant employs a standard key generation process, resulting in the expanded secret and public keys (epk,esk). The `pkc` (public key compact) variant introduces a compact representation for the public key (cpk), significantly reducing memory requirements (Figure 1). The ExpandPK algorithm is invoked during the verification phase to expand the public key before it can be used in verification computations, at the cost of increased latency (Fig. 2). Finally, the `pkc+skc` (public & secret key compact) variant further optimizes storage by employing compact representations for both the secret and public keys (csk,cpk). In addition to the modifications during the verification phase, the ExpandSK algorithm is executed during the signature generation phase to expand the secret key. This variant minimizes storage overhead at the expense of increased computation time during both signature generation and verification. We present the parameter set of the different variants of UOV in Table 2. Throughout this text, we will denote vector/matrix dimension $n-m$ as l.

`CompactKeyGen()`

- $\text{seed}_{\text{sk}} \leftarrow \{0,1\}^{\text{sk\_seed\_len}}$ ## sk_seed_len=256
- $(\text{seed}_{\text{pk}}, \mathbf{O}) := \text{Expand}_{\text{sk}}(\text{seed}_{\text{sk}})$ ## pk_seed_len=128
- $\left\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}\right\}_{i\in[m]} := \text{Expand}_{\mathbf{P}}(\text{seed}_{\text{pk}})$
- **for** $i=1$ upto m **do**
  - $\mathbf{P}_i^{(3)} := \text{Upper}\left(-\mathbf{O}^\top \mathbf{P}_i^{(1)} \mathbf{O} - \mathbf{O}^\top \mathbf{P}_i^{(2)}\right)$
- $cpk := \left(\text{seed}_{\text{pk}}, \left\{\mathbf{P}_i^{(3)}\right\}_{i\in[m]}\right)$
- $csk := \text{seed}_{\text{sk}}$
- **return** (cpk, csk)

`Sign(`esk,μ`)`

- $\text{salt} \leftarrow \{0,1\}^{\text{salt\_len}}$ ## salt_len=128
- $\mathbf{t} := \text{Hash}(\mu||\text{salt})$
- **for** $ctr=0$ upto 255 **do**
  - $\mathbf{v} := \text{Expand}_{\mathbf{v}}(\mu||\text{salt}||\text{seed}_{\text{sk}}||\text{ctr})$
  - $\mathbf{L} := \mathbf{0}_{m\times m}$
  - **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**
    - $\mathbf{s} := \begin{bmatrix}\mathbf{v}\\ \mathbf{0}_m\end{bmatrix} + \begin{bmatrix}\mathbf{O}\\ \mathbf{I}_m\end{bmatrix}\mathbf{x}$
    - $\sigma := (\mathbf{s}, \text{salt})$
    - **return** σ
- **return** $\perp$

`Verify(`epk,μ,σ`)`

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

Figure 1: Main UOV-DSA (pkc) routines [11]

`ExpandSK(`csk`)`

- $(\text{seed}_{\text{pk}}, \mathbf{O}) := \text{Expand}_{\text{sk}}(\text{seed}_{\text{sk}})$
- $\left\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}\right\}_{i\in[m]} := \text{Expand}_{\mathbf{P}}(\text{seed}_{\text{pk}})$
- **for** $i=1$ upto m **do**
  - $\mathbf{S}_i := \left(\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top}\right)\mathbf{O} + \mathbf{P}_i^{(2)}$
- $esk := \left(\text{seed}_{\text{sk}}, \mathbf{O}, \left\{\mathbf{P}_i^{(1)}, \mathbf{S}_i\right\}_{i\in[m]}\right)$
- **return** esk

`ExpandPK(`cpk`)`

- $\left\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}\right\}_{i\in[m]} := \text{Expand}_{\mathbf{P}}(\text{seed}_{\text{pk}})$
- **for** $i=1$ upto m **do**
  - $\mathbf{P}_i = \begin{bmatrix}\mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)}\\ \mathbf{0} & \mathbf{P}_i^{(3)}\end{bmatrix}$
- $epk := \{\mathbf{P}_i\}_{i\in[m]}$
- **return** epk

Figure 2: UOV-DSA secret- and public-key expansion [11]

3 SENSITIVITY ANALYSIS OF UOV-DSA

Performing a sensitivity analysis is a crucial first step in order to determine which variables require masking or protection to mitigate SCAs. We identify ExpandSK, CompactKeyGen, and Sign (Fig. 3 - 5) as vulnerable to differential power attacks, as they involve the secret key. They contain colour-coded representations of the sensitive components within the UOV scheme. 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. In contrast, both ExpandPK and

Table 2: Parameter sets for all UOV-DSA variants.

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

`Verify` are not sensitive, as they operate exclusively on public inputs and variables that do not facilitate signature forgery or key recovery.

`UOV.CompactKeyGen`. The compact key generation algorithm first samples the seed seed_{sk}, from which seed_{pk} and secret matrix $\mathbf{O}$, which corresponds to the oil space, are derived. Here seed_{sk} is a sensitive variable and $\text{Expand}_{\text{sk}}$ is a sensitive algorithm. Both need protection and are targets for applying masking, while the public key seed seed_{pk} can be unmasked after generation.

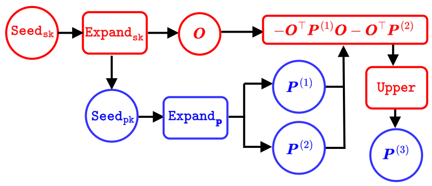


Figure 3: Sensitivity analysis of `UOV.CompactKeyGen`.

The seed_{pk} is used as input in $\text{Expand}_{\mathbf{P}}$ to construct two public matrices: $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(2)}$, which are used to compute $\mathbf{P}_i^{(3)}$. Here, seed_{pk}, $\mathbf{P}_i^{(1)}$, $\mathbf{P}_i^{(2)}$ and $\mathbf{P}_i^{(3)}$ are non- sensitive. However, the computation of $\mathbf{P}_i^{(3)}$ is a sensitive operation due to its dependence on the secret vector $\mathbf{O}$, as illustrated in the lower portion of the Figure 3. It is crucial to emphasize that, despite this dependence, the (final) value of $\mathbf{P}_i^{(3)}$ does not leak any information regarding the secret vector $\mathbf{O}$.

`UOV.ExpandSK`. The algorithm `ExpandSK` operates on the secret key, so it is sensitive and requires protection. As shown in Figure 4, the computation of $\mathbf{S}_i$ involves the sensitive variable $\mathbf{O}$. An adversary could derive the secret $\mathbf{O}$ from the matrices $\mathbf{S}_i$ and hence it requires masking. The variables $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(2)}$ are public and not sensitive.

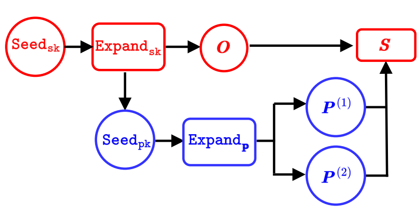


Figure 4: Sensitivity analysis of `UOV.ExpandSK`.

`UOV.Sign`. The signature generation algorithm takes the expanded secret key *esk* and message μ as input. It first samples a random salt, and computes the message digest $\mathbf{t}$ as $\text{Hash}(\mu||\text{salt})$. These variables are public and non-sensitive. 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 $\mathbf{v} \in \mathbb{F}_q^n$. This sampling can be done by running Expand_{v} on input seed_{sk},μ, and salt. This computation is sensitive, since it may leak information about seed_{sk}, and it may influence the distribution of $\mathbf{v}$ (Figure 5). Subsequently, $\mathbf{y} = [\mathbf{v}^{\top}\mathbf{P}_i^{(1)}\mathbf{v}]_{i\in[m]}$ is computed. This operation (and its variables) are sensitive because the adversary can perform a SCA to retrieve $\mathbf{v}$, which leads to a key recovery.

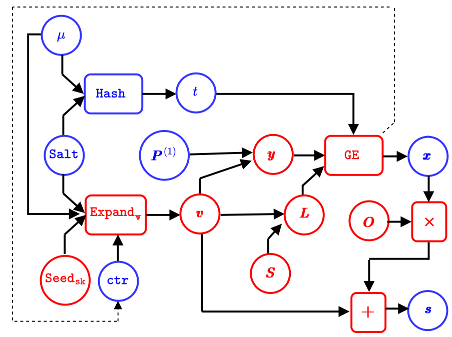


Figure 5: Sensitivity analysis of `UOV.Sign`.

Once the vinegar variable is fixed, then the quadratic system $\mathcal{P}(\mathbf{s}) = \mathbf{t}$ is converted to a linear system $\mathbf{Lx} = \mathbf{t} - \mathbf{y}$. Clearly $\mathbf{L}$ is sensitive and should be masked, as it can lead back to the oil space. Now, if $\mathbf{L}$ is invertible, then $\mathbf{x}$ can be computed by performing Gaussian elimination (GE), allowing the computation of $\mathbf{s}$, finally. Otherwise, $\mathbf{v}$ is re-sampled and the previous process is repeated. The GE operation must be masked because it can reveal information about the secret oil space $\boldsymbol{O}$. However, $\mathbf{x}$ is part of the public signature $\mathbf{s}$ and can thus be revealed after its computation. The execution time of signature generation leaks *ctr* value, so we can consider *ctr* is also non-sensitive.

Differences with [2]. While this work was ongoing, a similar work appeared on IACR ePrint and was subsequently published in PQCrypto '25. As the contents in our work differ from several claims and conclusions related to masking in the other work[2, Section 5.5], we explicitly go over those differences.

Protecting $\mathbf{S}_i$ and $\mathbf{O}$. First, we would like to explicitly highlight the need for applying the masking countermeasure on `CompactKeyGen` and `ExpandSK` routines to prevent DPAs in embedded environments. In such a scenario, the matrices $\mathbf{S}_i$ and $\mathbf{O}$ will be generated in a shared manner, as it involves the secret oilspace.

Protecting $\mathbf{v}$. Second, the authors of [2] propose to unmask the sensitive variable $\mathbf{v}$ during the quadratic evaluation and computation of $\mathbf{y} = [\mathbf{v}^{\top}\mathbf{P}_i^{(1)}\mathbf{v}]_{i\in[m]}$. Instead, they propose to mask the public values $\mathbf{P}_i^{(1)}$. The main argument provided by the authors is an SPA [3]. As this matrix is public, it does not require side-channel protection and unmasking the sensitive value exposes an implementation against DPAs. Additionally, masking is not an appropriate technique for protecting against SPAs. Instead, our approach protects against any DPA and alternative countermeasures can be integrated to protect against SPAs. This includes shuffling the operations in the multiplications with the public values and performing a mask refreshing on $\mathbf{v}$, between the multiplication of each share with $\mathbf{P}_i^{(1)}$. In this case, if an attacker is able to obtain one share of $(\mathbf{v}_i)$ via a profiling attack, the next share will be randomly masked via a different random. As such, even if an attacker obtains all shares, the original $(\mathbf{v}_i)$ can never be reconstructed as all shares belong to a different set of fresh masks. We note that all multiplications of sensitive variables with a public value are vulnerable against such SPAs and require appropriate hardening.

Protecting L. In contrast to the claims in prior work, the sensitive matrix **L** requires protection against DPAs (through masking). This includes solving the system of linear equations $\mathbf{Lx} = \mathbf{t} - \mathbf{y}$ through masked gaussian elimination, instead of the unmasked variant proposed in [2].

4 EFFICIENT MASKED GADGETS

In this section we propose and introduce masking techniques for all sub-operations in the UOV DSA. All novel gadgets are described by a t-order algorithm ($n = t+1$ shares) and accompanied with a detailed description. More specifically:

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

All components, including gadgets from literature, required to achieve fully masked UOV (Section 5) are listed in Table 3.

Table 3: Overview of used gadgets, with $n = t+1$ shares.

Algorithm	Description	Security	Reference
`SecREF`	Refresh of Boolean masking	t-SNI	[6, 17]
`FullAdd`	Secure unmasking of Boolean shares	t-NI	[7, 16] & 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	[37] & Alg. 8
`SecBackSub`	Masked back substitution with public output	t-NIo	[37] & Alg. 9

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. Additionally, we mechanically verify all (first- and second-order) t-(S)NI claims/proofs using the verification tool `maskVerif`.

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 typical 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 naïvely thus requires performing l masked multiplications (i.e. re-sharing and compression) and summing the l results. We propose a more efficient technique: 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 [26], modifying the domain-oriented ISW multiplication [27, 28] by delaying the compression stage when chaining multiplications.

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**
    $r_{ij} \leftarrow \mathbb{F}_q$
    $u_{ij} = u_{ij} + r_{ij}$
    $u_{ji} = u_{ji} + r_{ij}$
$(z_i) := (w_i + \sum_{j=1, j\neq i}^{n} u_{ij})$ `## Compression`
return (z_i)

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.

Compression. The refreshed partial sums are now combined into the final output values z_i. As proposed in [22], 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. Here, we discuss the run-time complexity (number of operations) and randomness complexity of the `SecDotProd` operation, following the approach proposed in [15, 43]. We denote the run-time and randomness complexity of an operation `Operation` by $T_{\texttt{Operation}}$ and $R_{\texttt{Operation}}$, respectively. We also assume that the run-time cost of random number generation is unit time and operands are $w = \lceil \log(q) \rceil$ bits wide. The run-time and randomness complexity of `SecDotProd` are:

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

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 B. Additionally, we verify its t-SNI notion using maskVerif, for first- and second-order.

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:

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

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. □

Additionally, we verified that the SecMatVec gadget satisfies the t-SNI notion at first- and second-order using maskVerif.

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 $\mathbf{T} = \{\mathbf{P}_j\}_{j\in[m]}\mathbf{x}$. 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 $(\mathbf{s}_i) :=$ SecREF$((\mathbf{x}_i))$
2 **for** $j = 1$ *upto* m **do**
3   $(\mathbf{T}[:,j]_i) = (\mathbf{P}_j\mathbf{x}_i)$ /* $\mathbf{T}_i \in \mathbb{F}_q^{l\times m}$ */
4 $(\mathbf{y}_i) :=$ SecMatVec$((\mathbf{T}_i^T),(\mathbf{s}_i))$ /* $\mathbf{y}^T = (\mathbf{x}^T\mathbf{T})^T = \mathbf{T}^T\mathbf{x}$ */
5 **return** $(\mathbf{y}_i)$

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

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

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 9c depicts an overview of the construction of Algorithm 3 from its elementary gadgets. Apart from those listed in Table 3, 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

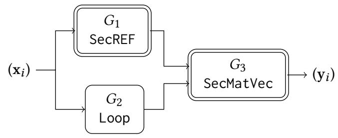


Figure 6: An abstract diagram of `SecQuad` (Algorithm 3). The t-NI gadgets are depicted with a single border, the t-SNI gadgets with a double border.

(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. □

Finally, we mechanically verify that the `SecQuad` gadget satisfies the t-SNI notion at first- and second-order, using `maskVerif`.

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 [17]. 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 [17] that the t-SNI refresh in [6] also satisfies the free-t-SNI notion. We refer to [17, 18] 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 [37]. We recall the `SecRowEch` and `SecBackSub` gadgets in Appendix C. 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. Additionally, we integrate the early stop during the initial phase of masked gaussian elimination to improve performance, as proposed in the UOV specification [11]. As a result, only a few instead of all rows are conditionally added to the pivot row in an attempt to make it non-zero.

5 MASKING UOV AT ARBITRARY ORDER

In this section, we combine the different masked gadgets described in Sec. 4 to design masked components of UOV [11]. 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.

5.1 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.

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 $\mathrm{seed}_{\mathrm{sk}}$ returned as a Boolean sharing. Each share is a randomly sampled binary string of length sk_seed_len. Since the Round 2 specification, the public seed is derived from the (shared) secret key seed. 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 $cpk = (seed_{pk}, \left\{\mathbf{P}_j^{(3)}\right\}_{j\in[m]})$. This procedure is explained below.

Algorithm 4: `mCompactKeyGen`

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

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

Generation of O. The shares of the secret matrix **O** are obtained by expanding the masked seed $((\mathrm{seed}_{\mathrm{sk},i})_{1\leq i\leq n})$ using the masked PRNG `mExpand`$_{\mathrm{sk}}$ in Line 2. The masked PRNG is instantiated using masked `shake256()`, derived from the Keccak primitive, and produces Boolean shares $(\mathbf{O}_i)$. Additionally, the public key seed $\mathrm{seed}_{\mathrm{pk}}$ is derived in this step, but can be unmasked and made public after its generation.

Computation of $\{\mathbf{A}_j\}_{j\in[m]} = \left\{-\mathbf{P}_j^{(1)}\mathbf{O}-\mathbf{P}_j^{(2)}\right\}_{j\in[m]}$. 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 3), and are used to compute the third $\left\{\mathbf{P}_j^{(3)}\right\}_{j\in[m]}$. The first step is to compute m matrices $\{\mathbf{A}_j\}_{j\in[m]}$ in a masked fashion (Line 6). 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 7).

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 9, 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 $\left\{\mathbf{P}_j^{(3)}\right\}_{j\in[m]}$. The `Upper` function is applied share by share, on each of m matrices $\{\mathbf{B}_j\}$ in Line 10. Finally, one can securely recombine the shares of each $\mathbf{B}_j$ to obtain each $\mathbf{P}_j^{(3)}$, using the `FullAdd` gadget (Line 11). Its details are discussed in Section 4.4 and its security in a larger composition is explained below.

5.1.1 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 $\left\{\mathbf{P}_j^{(3)}\right\}_{j\in[m]}$, 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 5.1. *The gadget* `mCompactKeyGen` *(Algorithm 4) is t-NIo secure with public output $\left\{\mathbf{P}_j^{(3)}\right\}_{j\in[m]}$.*

Proof. We model a single iteration j of Algorithm 4 as a sequence of t-(S)NI gadgets, which is visually shown in Figure 7. In addition to the gadgets listed in Table 3, we model the linear operations in Line 6-7 and Line 10 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 $\mathtt{mExpand}_{\mathtt{sk}}$, which produces a uniformly masked matrix $\mathbf{O}$. We also consider the iterations of the loop in Line 8-9 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 7, 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 $t_{A_4} = \sum_{i=1}^{4} t_{G_i} + \sum_{i=1}^{3} o_{G_i}, \quad |O| = o_{G_4}$.

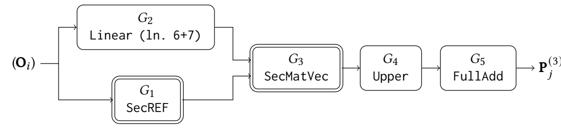


Figure 7: An abstract diagram of an iteration j in `mCompactKeyGen` (Alg. 4). The t-NI gadgets are depicted with a single border, the t-SNI gadgets with a double border.

We now prove Part I of Lemma 5.1 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 $\left\{\mathbf{P}_j^{(3)}\right\}_{j\in[m]}$. ◻

We verify the t-SNI property (Part I) of gadgets G_1-G_4 using `maskVerif`, at first- and second-order. Due to the tool's limitation with handling the NIo notion, we are not able to mechanically verify the full composition in Algorithm 4.

5.2 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. Our strategy consists of using the shared compact secret key to generate the shared expanded key in a masked fashion.

Again, the sensitive secret key *csk* contains the Boolean masked $(seed_{sk,i})_{1\leq i\leq n}$. It is used to compute the masked expanded secret key components: matrix $(\mathbf{O}_i)$ and matrices $\{(\mathbf{S}_{j,i})_{1\leq i\leq n}\}_{j\in[m]}$.

Generation of $\mathbf{O}$ and $\{\mathbf{P}_j^{(1)},\mathbf{P}_j^{(2)}\}_{j\in[m]}$. We refer to Section 5.1, as this procedure (Line 1 - 2) is identical in `mCompactKeyGen`.

Computation of $\{\mathbf{S}_j\}_{j\in[m]} = \left\{\left(\mathbf{P}_j^{(1)}+\mathbf{P}_j^{(1)T}\right)\mathbf{O}+\mathbf{P}_j^{(2)}\right\}_{j\in[m]}$. The sequence of matrices $\{\mathbf{S}_j\}_{j\in[m]}$ 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).

Algorithm 5: `mExpandSK`

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

$(\text{seed}_{\text{pk}}, (\mathbf{O}_i)) := \texttt{mExpand}_{\text{sk}}((\text{seed}_{\text{sk},i}))$
$\{\mathbf{P}_j^{(1)}, \mathbf{P}_j^{(2)}\}_{j\in[m]} := \texttt{Expand}_{\mathbf{P}}(\text{seed}_{\text{pk}})$
for $j = 1$ *upto* m **do**
  $(\mathbf{S}_{j,i})_{1\le i\le n} := \left(\left(\mathbf{P}_j^{(1)} + \mathbf{P}_j^{(1)T}\right)\mathbf{O}_i\right)$ /* $\mathbf{S}_{j,i} \in \mathbb{F}_q^{l\times m}$ */
  $\mathbf{S}_{j,1} = \mathbf{S}_{j,1} + \mathbf{P}_j^{(2)}$
return $esk = ((\text{seed}_{\text{sk},i}), (\mathbf{O}_i), \{\mathbf{P}_j^{(1)}, (\mathbf{S}_{j,i})_{1\le i\le n}\}_{j\in[m]})$

5.2.1 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 5.2. *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 $(\mathbf{O}_i)$. □

The composition of Algorithm 5 is mechanically verified to be t-NI secure at first- and second-order, using `maskVerif`.

5.3 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.

Following its expansion (see previous section), the expanded secret key consists of three Boolean shared components: $(\text{seed}_{\text{sk},i})$, $(\mathbf{O}_i)$ and $\{(\mathbf{S}_{j,i})_{1\le i\le n}\}_{j\in[m]}$. The secret $(\text{seed}_{\text{sk},i})$ 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 $\sigma = (\mathbf{s}, \text{salt})$.

Generation of v. The shares of the secret vinegar vector $\mathbf{v}$ are sampled from a masked PRNG $\texttt{mExpand}_{\mathbf{v}}$ in Line 4, based on the message μ, the masked secret seed $(\text{seed}_{\text{sk},i})$, 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 $(\mathbf{y}_i)$.

Algorithm 6: `mSign`

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

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

Solving $\mathbf{L}\mathbf{x} = \mathbf{t} - \mathbf{y}$. The system of linear equations is solved using masked Gaussian elimination, using the techniques introduced in [37]. 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).

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 $\mathbf{s}$.

5.3.1 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 $\mathbf{s}$ and $\mathbf{c}$. The signature $\mathbf{s}$ is public, while $\mathbf{c}$ 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 5.3. *The gadget* `mSign` *(Algorithm 6) is t-NIo secure with public outputs* $\mathbf{s}$ $(\mathbf{w}, \mathbf{x})$ *and* $\mathbf{c}$.

Proof. We model a single iteration of Algorithm 6 as a composition of t-(S)NI gadgets, which is visually shown in Figure 8. Apart from the gadgets listed in Table 3, 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

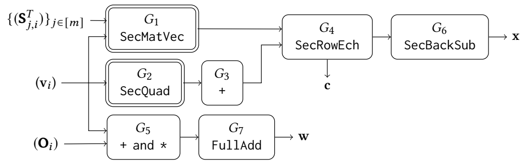


Figure 8: An abstract diagram of an iteration ctr in `mSign` (Alg. 6). The t-NI gadgets are depicted with a single border, the t-SNI gadgets with a double border.

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, the 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 `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 $\mathbf{x}$,$\mathbf{w}$ and $\mathbf{c}$. 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 $\mathbf{c}$. 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 $\mathbf{s}$ and $\mathbf{c}$.

Finally, we note that the signing procedure only requires multiple iterations if the system of linear equations is unsolvable and no unique solution $\mathbf{x}$ 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 $\mathbf{s}$ and $\mathbf{c}$. □

We verify the t-SNI property of the composition of gadgets G_1, G_2 & G_5 using `maskVerif`, at first- and second-order. Due to the tool's limitation with handling the NIo notion, we are not able to mechanically verify the full composition in Algorithm 6.

6 IMPLEMENTATION AND EVALUATION

We complement our theoretical analysis from Section 4 & 5 with a practical side-channel leakage evaluation and performance analysis of our masked implementation of UOV.

6.1 Practical Security Evaluation

In this section we complement our theoretical and formal security analysis of the proposed gadgets with a practical side-channel leakage evaluation on a real-world device. We confirm the theoretical results of the previous sections and demonstrate how our proposed techniques lead to efficient and practically secure implementations on real-world devices.

6.1.1 Micro-Architectural Effects & Compiler Optimization. To ensure our implementation is secure, we take explicit measures to mitigate side-channel leakage due to micro-architectural effects, such as transitional leakage in memory elements. This effect leads to unexpected leakages and is a result of successively writing both shares (r and $x \oplus r$) of a (first-order masked) sensitive variable x into one (pipeline) register or ALU unit. This will cause the power consumption of the device to be correlated to the original secret x, as $\mathrm{HD}(x \oplus r, r) = \mathrm{HW}(x)$. We carefully craft and deploy various *clearing* routines, written in assembly, to mitigate leakages due to micro-architectural effects. They pre-load affected memory elements with a random value, before loading the second share, to break such secret dependencies.

Prior work typically relies on turning off compiler optimizations (`-o0`) to ensure masking countermeasures (written in C) are not optimized away [4, 52], at the cost of performance degradation. For example, allowing (aggressive) compiler optimizations might result in the removal of *intentional* dummy-operations or re-ordering of instructions. However, as demonstrated below, we find our implementation remains secure when using compiler optimization flag `-o3`, allowing for high optimization. By ensuring the compiler does not re-order any (security-critical) operations through the compiler flags `-fno-schedule-insns` and `-fno-schedule-insns2`, the resulting machine code is efficient and remains secure.

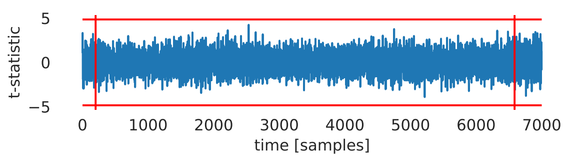
6.1.2 Measurement Setup. We conduct our measurements from the dedicated measurement port on a NewAE CW308 UFO board with an `STM32F415RG` Arm Cortex-M4[1] microcontroller as target. We supply the target board with an external 8 MHz clock and configure it to internally run at a 24 MHz operating frequency. For trace acquisition, we use a 6426E PicoScope with 12-bit resolution and the sampling rate set to 125 MS/s. We synchronize the oscilloscope with the external clock during all measurements and use the on-chip TRNG for on-the-fly randomness generation. The micro-controller sends a trigger signal before (and after) the target operation, indicated by vertical red lines in each figure below. The initial sharings are computed randomly and sent directly to the micro-controller. As is common practice, we choose a reduced (UOV) parameter set ($m = 4$, $n = 10$) for our practical security evaluation to ensure practical feasibility. The operations performed in all gadgets are not impacted by the choice of m or n and only the amount of iterations are.

6.1.3 TVLA Methodology. The side-channel security of our masked implementation is evaluated using the Test Vector Leakage Assessment (TVLA) methodology [25] and the *non-specific, fixed vs. random* t-test statistic. More specifically, power measurements are taken from the target device which is operating on either a fixed or random input, in a random fashion. Subsequently, two sets of power traces are constructed: S_f and S_r. Finally, the Welch's two-tailed t-test is computed for each sample point in the trace to determine if the masking countermeasure is secure. If the t-value doesn't exceed a threshold value, typically ± 4.5, the samples from both sets cannot be distinguished with high confidence ($\alpha = 0.01$), i.e. no leakage. We recall that this threshold value is not universal, as computing

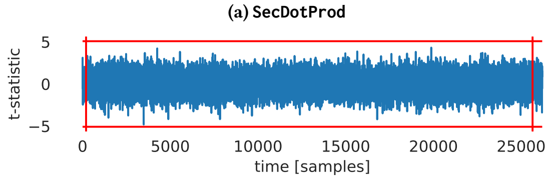
[1] arm-none-eabi-gcc 10.3.1 20210621

the t-value on many samples (long traces) will, with high probability, introduce false positives which exceed ±4.5 (see [21, Table 1] and [4, Appendix A]). Below, we adapt the threshold value for each experiment accordingly (Appendix E).

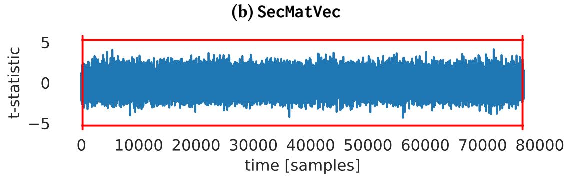
6.1.4 Results. The first-order TVLA results with RNG ON for the `SecDotProd`, `SecMatVec` and `SecQuad` gadgets are shown in Figure 9. The results confirm our theoretical expectation, as the t-value does not cross the threshold value after acquiring one million traces and thus the implementations under test are considered secure. Additionally, we include the mean power traces and first-order statistical moments with the masking countermeasure disabled (randomness set to zero) in Appendix E (Fig. 12 - 14). For `SecDotProd`, significant leakages can be detected with only 50 thousand traces, guaranteeing the soundness of our set-up.



(a) `SecDotProd`



(b) `SecMatVec`

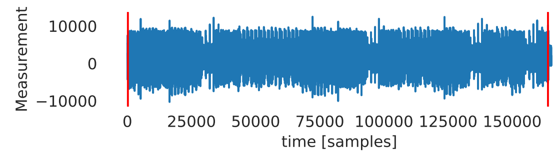


(c) `SecQuad`

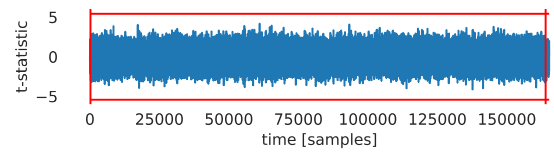
Figure 9: TVLA analysis of 1st-order `SecDotProd`, `SecMatVec` & `SecQuad` (RNG ON) with 1M traces. The t-test threshold is marked by red lines.

Additionally, we adapt the implementation of [37] for solving the system of linear equations in the masked signing procedure, with early stop (Section 4.4). Figure 10 shows the first-order TVLA results of masked Gaussian elimination. We successfully apply our methodology to remove leakages due to micro-architectural effects and do not allow the compiler to re-order instructions. As can be seen, no leakage is detected after acquiring 100K traces, while significant leakages are detected with our countermeasure turned off after 1K traces (Fig. 10c).

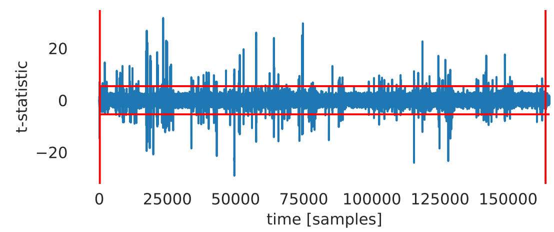
In summary, we have successfully demonstrated how our masked gadgets (Section 4 & 5) result in efficient and first-order secure implementations on the Arm Cortex-M4. We find minimal performance overhead can be achieved through enabling compiler optimizations, without impacting side-channel security in practice. We leave the investigating of the applicability of aggressive compiler optimization (`-o3`) without instruction re-ordering to other masked implementations as future work.



(a) Mean Trace



(b) RNG ON



(c) RNG OFF

Figure 10: TVLA analysis of 1st-order `SecRowEch` [37]. Mean trace and t-test results with RNG ON (100K traces) and RNG OFF (1K traces). The ± 5.423 threshold is marked by red lines.

6.2 Performance Evaluation and Comparison

This section presents the implementation results of masked UOV algorithms and compares them with the state-of-the-art post-quantum masked digital signature algorithms. We have used a DELL Latitude E7470 laptop with an Intel (R) Core (TM) i7-6600 CPU running at 2.60 GHz and the GCC 6.5 compiler with optimization flag 03 to calculate the performance of our algorithms in cpucycle counts.

The performance of our masked key generation and masked signature generation of the UOV schemes is presented in Table 4. We have implemented and documented the performance results for the UOV parameter sets UOV-Ip and UOV-III, the pkc variant. However, our code can be extended to other variants of the UOV scheme. The performance overhead factor in the masked key generation of UOV-Ip for the first-, second-, and third-order are 13×, 38×, and 62×, respectively. The masked signature generation algorithm of UOV-Ip has 12×, 48×, and 78× overhead for first-, second-, and third-order masking. These overhead factors for the masked algorithms of UOV-III are similar to or slightly larger than those of UOV-Ip.

We also compare the performance result of the masked signature generation of UOV-Ip with the masked signature generation of Dilithium and Falcon in Table 4. We also compare the performance result of the masked signature generation of UOV-Ip with the state-of-the-art masked signature generation of Dilithium [14] and Falcon [8] in Table 4. Although the unmasked signature generation of the NIST standard Dilithium2 and UOV-Ip requires a similar amount of CPU cycles, the first-order, second-order, and third-order masked signature generation of UOV-Ip perform, respectively, 62%, 19%, and 25% better than Dilithium2. For easier visualization, we

Table 4: Comparing the performance of masked implementations of UOV with the state-of-the-art PQ masked digital signature schemes. The performances are in 1000× cpucycles unless otherwise mentioned.

Scheme	Key-generation (1000× cpucycles)				Sign (1000× cpucycles or seconds)			
	unmasked	Masked			unmasked	Masked		
		1st	2nd	3rd		1st	2nd	3rd
UOV-Ip [TW]	58,772	**791,871 (13×)**	**2,215,190 (38×)**	**36,616,77 (62×)**	1,268	**15,490 (12×)**	**61,420 (48×)**	**99,469 (78×)**
Dilithium2 [14]	-	-	-	-	1,228	40,368 (33×)	76,173 (62×)	133,508 (109×)
Falcon 512* [8]	-	-	-	-	-	3.199 s	6.368 s	-
UOV-III [TW]	370,483	**5,670,203 (15×)**	**16,241,454 (44×)**	**25,509,566 (69×)**	5,064	**62,110 (12×)**	**259,969 (51×)**	**415,293 (82×)**

*: The performances are in seconds as the processor frequency was not provided. Except this, all others are measured using the same laptop.

compare the execution time (in seconds) of first-, second-, and third-order masked signature generations of UOV-Ip with Dilithium2 in Figure 11. The masked signature generation of UOV compared to Falcon 512 performs 99.8% better for first-order and 99.6% better for second-order masking.

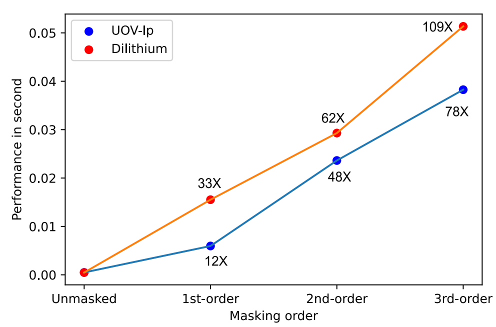


Figure 11: Comparison of masked UOV-Ip with Dilithium2

Finally, we also want to note that our primary objective in this work was to design the masking algorithms and provide theoretical and empirical proofs for security. Our implementations can be considered as a small improvement over proof-of-concept and not a fully optimized version. Therefore the efficiency can be improved further. This is in contrast to ML-DSA, which, due to its popularity, has gone through a larger cycle of optimizations and improvements. Further, 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.

7 IMPACT AND DISCUSSIONS

Resource-constrained devices such as IoT, sensor nodes, medical monitoring devices, etc., usually operate under very stringent operational constraints [31]. Throughout this work, we shed light on another important aspect of deploying PQC in the real world, especially on RCD: masking PQ DSA schemes. Due to the ubiquitous nature of RCDs, they are often easy targets for SCA adversaries. One such compromised device can jeopardize the security of the whole network. Therefore, from the perspective of widespread deployment, it is crucial to protect these devices from such adversaries. However, due to their limited resources, it is also difficult to incorporate sophisticated and strong SCA countermeasures.

However, the public key (verification key) is quite large for UOV and its MQE-based variants. As mentioned in Section 1, for the Round 2 specification of UOV [11], the signature size is 96 bytes, yet the verification key is approximately 66 KB. Therefore, UOV is particularly useful where the public key does not need to be transmitted, or in a hybrid with other PQ DSAs. In many cases, public keys can be pre-distributed to reduce the overhead as the work [45] suggests. Alternatively, following the example (TLS handshake) given in [50], a minimum of two public keys (public key of intermediate and terminal server) need to be transmitted along with five signatures (certificates of root, intermediate, and leaf certificates along with two signed certificates timestamps according to current standards), assuming only one intermediate certificate. Since root servers and signed certificate timestamps do not carry their public keys during handshake, one can use UOV DSA there, and ML-DSA can be used in the rest of the fields. This reduces the required bandwidth during the TLS handshake to 7.2 KB from 14.7 KB if ML-DSA is used everywhere.

In constrained radio networks, ephemeral Diffie-Hellman over COSE (EDHOC) [46] is a lightweight Diffie-Hellman key-exchange protocol used in RCD design by the Internet Engineering Task Force (IETF). In typical use cases it needs to send 3 messages of size 37, 45, and 19 bytes (total 101 bytes) using classical cryptography. If a combination of ML-KEM [35] with ML-DSA [33] is used, these sizes would be 773, 3194, and 2433 bytes or 6400 bytes in total. However, a hybrid of ML-KEM and UOV reduces the sizes of the messages to 773, 870, and 109 bytes or 1752 bytes in total. Although the large size of ML-KEM dominates the total size in the latter case it still offers approximately 3.6x improvement over the ML-KEM + ML-DSA hybrid. Another protocol used in CRN and also designed by IETF is Group OSCORE [48], which provides end-to-end security for group communications in RCD. Here, a non-interactive key-exchange (NIKE) is used. Therefore, the payload size is equal to the signature size in addition to some headers. In this protocol and others where the public key is not sent while sending the certificate and signing is performed on the RCD, using UOV DSA has almost similar overhead as compared to the classical signatures and a clear advantage over other PQ DSAs.

Our results show that in the context of masking on RCD, UOV is a far better alternative than ML-DSA and a strong candidate for easing PQ DSA migration. Although several works [20, 31, 38] mentioned that the energy cost for computations is less energy intensive than radio transmission, an overhead of (33-109×) for masked implementations is significant. UOV with smaller overhead has much better leeway to provide higher-order security for a similar cost in energy. It also leaves room to incorporate other types of countermeasures, such as duplication countermeasures, to prevent fault injection attacks.

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. Uttam Kumar Ojha worked on this work during his internship at COSIC. With utmost gratitude, Uttam Kumar Ojha thanks Professors Bart Preneel & Bimal K. Roy for the internship opportunity at COSIC. Anindya Ganguly is supported by TCS Research fellowship. The work of Angshuman Karmakar is partially supported by the Research-I foundation from Infosys, the Initiation grant from IIT Kanpur, the Google India research fellowship, and PM ECRG by ANRF, Govt. of India.

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, and Juliane Krämer. 2025. SoK: On the Physical Security of UOV-Based Signature Schemes. In *Post-Quantum Cryptography*, Ruben Niederhagen and Markku-Juhani O. Saarinen (Eds.). Springer Nature Switzerland, Cham, 199–231.

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

[4] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. 2014. On the Cost of Lazy Engineering for Masked Software Implementations. In *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers (Lecture Notes in Computer Science, Vol. 8968)*, Marc Joye and Amir Moradi (Eds.). Springer, 64–81. https://doi.org/10.1007/978-3-319-16763-3_5

[5] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and Francois-Xavier Standaert. 2019. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *Computer Security – ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I* (Luxembourg, Luxembourg). Springer-Verlag, Berlin, Heidelberg, 300–318. https://doi.org/10.1007/978-3-030-29959-0_15

[6] 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

[7] 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.

[8] Pierre-Augustin Berthet, Justine Paillet, Cédric Tavernier, Lilian Bossuet, and Brice Colombier. 2024. Masked Computation of the Floor Function and Its Application to the FALCON Signature. *IACR Commun. Cryptol.* 1, 4 (2024), 9. https://doi.org/10.62056/AY73ZL7S

[9] Ward Beullens. 2021. Improved Cryptanalysis of UOV and Rainbow. In *Advances in Cryptology – EUROCRYPT 2021*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer International Publishing, Cham, 348–373.

[10] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. 2025. MAYO Specification Document Version 2.0. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/mayo-spec-round2-web.pdf

[11] 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. 2025. UOV: Unbalanced Oil and Vinegar Algorithm Specifications and Supporting Documentation Version 2.0. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/uov-spec-round2-web.pdf

[12] 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.

[13] Benoît Cogliati, Pierre-Alain Fouque, Louis Goubin, and Brice Minaud. 2024. New Security Proofs and Techniques for Hash-and-Sign with Retry Signature Schemes. Cryptology ePrint Archive, Paper 2024/609. https://eprint.iacr.org/2024/609

[14] Jean-Sébastien Coron, François Gérard, Tancrède Lepoint, Matthias Trannoy, and Rina Zeitoun. 2024. Improved High-Order Masked Generation of Masking Vector and Rejection Sampling in Dilithium. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2024, 4 (2024), 335–354. https://doi.org/10.46586/TCHES.V2024.I4.335-354

[15] Jean-Sébastien Coron. 2017. High-Order Conversion from Boolean to Arithmetic Masking. In *CHES 2017 (LNCS, Vol. 10529)*, Wieland Fischer and Naofumi Homma (Eds.). Springer, Cham, 93–114. https://doi.org/10.1007/978-3-319-66787-4_5

[16] 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.

[17] 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

[18] 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

[19] 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.

[20] Giacomo de Meulenaer, François Gosset, François-Xavier Standaert, and Olivier Pereira. 2008. On the Energy Cost of Communication and Cryptography in Wireless Sensor Networks. In *2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. 580–585. https://doi.org/10.1109/WiMob.2008.16

[21] A. Adam Ding, Liwei Zhang, Francois Durvaux, Francois-Xavier Standaert, and Yunsi Fei. 2018. Towards Sound and Optimal Leakage Detection Procedure. In *Smart Card Research and Advanced Applications*, Thomas Eisenbarth and Yannick Teglia (Eds.). Springer International Publishing, Cham, 105–122.

[22] 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

[23] 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.

[24] Hiroki Furue, Yasuhiko Ikematsu, Fumitaka Hoshino, Tsuyoshi Takagi, Haruhisa Kosuge, Kimihiro Yamakoshi, Rika Akiyama, Satoshi Nakamura, Shingo Orihara, and Koha Kinjo. 2025. QR-UOV Specification Document. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/qruov-spec-round2-web.pdf

[25] Gilbert Goodwill, Benjamin Jun, Joshua Jaffe, and Pankaj Rohatgi. 2011. A testing methodology for side channel resistance. Online. Accessed 8th January, 2025. https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf

[26] 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

[27] 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

[28] 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.

[29] David S Johnson and Michael R Garey. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman.

[30] 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.

[31] John Preuß Mattsson, Göran Selander, Ben Smeets, and Erik Thormarker. 2022. Constrained Radio Networks, Small Ciphertexts, Signatures, and Non-Interactive Key Exchange. *Fourth PQC Standardization Conference* (2022), 10. https://csrc.nist.gov/csrc/media/Events/2022/fourth-pqc-standardization-conference/documents/papers/constrained-radio-networks-pqc2022.pdf

[32] 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

[33] 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

[34] 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

[35] NIST. 2024. Module-Lattice-Based Key-Encapsulation Mechanism Standard. Online. Accessed 10th April, 2025. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf
[36] 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
[37] Quinten Norga, Suparna Kundu, Uttam Kumar Ojha, Anindya Ganguly, Angshuman Karmakar, and Ingrid Verbauwhede. 2025. Masking Gaussian Elimination at Arbitrary Order with Application to Multivariate-and Code-Based PQC. In *Topics in Cryptology – CT-RSA 2025*, Arpita Patra (Ed.). Springer Nature Switzerland, Cham, 249–272.
[38] Markku-Juhani O. Saarinen. 2020. Mobile Energy Requirements of the Upcoming NIST Post-Quantum Cryptography Standards. In *2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. 23–30. https://doi.org/10.1109/MobileCloud48802.2020.00012
[39] 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
[40] Pierre Pébereau. 2024. One Vector tonbsp;Rule Them All: Key Recovery fromnbsp;One Vector innbsp;UOV Schemes. In *Post-Quantum Cryptography: 15th International Workshop, PQCrypto 2024, Oxford, UK, June 12–14, 2024, Proceedings, Part II* (Oxford, United Kingdom). Springer-Verlag, Berlin, Heidelberg, 92–108. https://doi.org/10.1007/978-3-031-62746-0_5
[41] Pierre Pébereau. 2024. Singular points of UOV and VOX. Cryptology ePrint Archive, Paper 2024/219. https://eprint.iacr.org/2024/219
[42] Quantum-Resistant Cryptography Research Group. 2025. KpqC Competition Round 2. https://www.kpqc.or.kr/competition_02.html Accessed: 2025-04-09.
[43] 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
[44] Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2020. Post-Quantum TLS Without Handshake Signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1461–1480. https://doi.org/10.1145/3372297.3423350
[45] Peter Schwabe, Douglas Stebila, and Thom Wiggers. 2021. More Efficient Post-quantum KEMTLS with Pre-distributed Public Keys. In *Computer Security – ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I* (Darmstadt, Germany). Springer-Verlag, Berlin, Heidelberg, 3–22. https://doi.org/10.1007/978-3-030-88418-5_1
[46] Göran Selander, John Preuß Mattsson, and Francesca Palombini. 2024. Ephemeral Diffie-Hellman Over COSE (EDHOC). RFC 9528. https://doi.org/10.17487/RFC9528
[47] 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
[48] Marco Tiloca, Göran Selander, Francesca Palombini, John Preuß Mattsson, and Jiye Park. 2022. *Group OSCORE - Secure Group Communication for CoAP*. Internet-Draft draft-ietf-core-oscore-groupcomm-14. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-core-oscore-groupcomm/14/ Work in Progress.
[49] Lih-Chung Wang, Chun-Yen Chou, Jintai Ding, Yen-Liang Kuan, Jan Adriaan Leegwater, Ming-Siou Li, Bo-Shu Tseng, Po-En Tseng, and Chia-Chun Wang. 2025. SNOVA Specification Document Version 2.0. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/snova-spec-round2-web.pdf
[50] Bas Westerbaan and Luke Valenta. 2024. A look at the latest post-quantum signature standardization candidates. online. https://blog.cloudflare.com/another-look-at-pq-signatures/.
[51] 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
[52] Jannik Zeitschner and Amir Moradi. 2024. PoMMES: Prevention of Micro-architectural Leakages in Masked Embedded Software. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024, 3 (Jul. 2024), 342–376. https://doi.org/10.46586/tches.v2024.i3.342-376

APPENDICES

A Artifact & Data Availabiltiy

We make the source-code of our arbitrary-order masked implementation of UOV available, which we plan to open-source. Additionally, we make the scripts for formally verifying the security notions of our proposed gadgets at first- and second-order using `maskVerif` available.

B 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 uji 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. □

C Auxiliary Algorithms

Algorithm 7: FullAdd, from [7, 16]

Data: A Boolean sharing (y_i)
Result: Unmasked value y such that $y=\sum_{i=1}^{n} y_i$

$(a_i) := \texttt{SecREF}((y_i))$ /* free-t-SNI */
$y := a_1+\cdots+a_n$
return y

Algorithm 8: SecRowEch, from [37]

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 ⊥

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

Algorithm 9: SecBackSub, from [37]

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 $\mathbf{Ax}=\mathbf{b}$

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

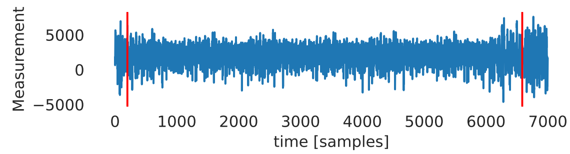
D Complexity Algorithm 6 (`mSign()`)

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

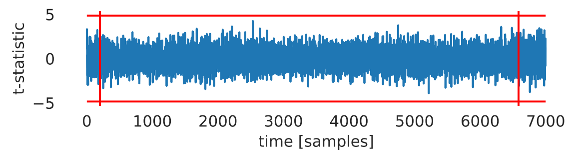
$$
\begin{aligned}
T_{\texttt{mSign}}(l,m,n) = {} & 1+T_{\texttt{Hash}}(m,n)+\texttt{ctr}\cdot(T_{\texttt{mExpand}_{\boldsymbol{v}}}(l,n) \\
& +(m\cdot T_{\texttt{SecMatVec}}(l,m,n))+(T_{\texttt{SecQuad}}(l,m,n)) \\
& +(m)+(T_{\texttt{SecRowEch}}(m,n))+(T_{\texttt{SecBackSub}}(m,n)) \\
& +(lmn+ln)+(l\cdot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

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

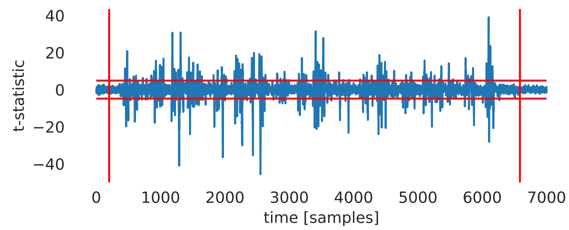
E Other TVLA Results



(a) Mean Trace

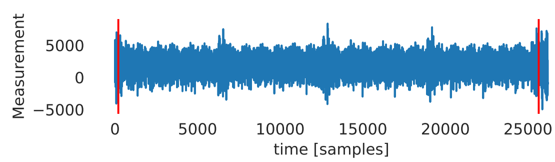


(b) RNG ON

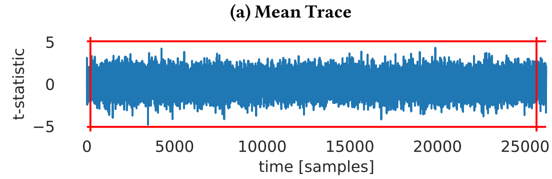


(c) RNG OFF

Figure 12: TVLA analysis of 1st-order `SecDotProd`. Mean trace and t-test results with RNG ON (1M traces) and RNG OFF (50K traces). The ± 4.849 threshold is marked by red lines.

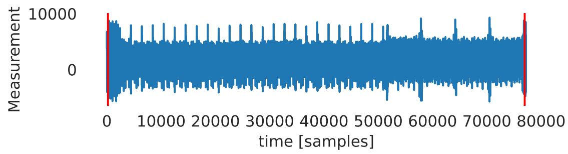


(a) Mean Trace

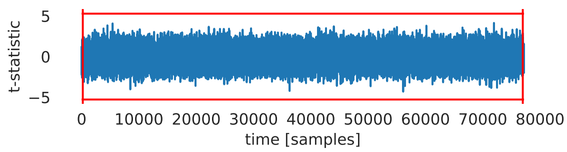


(b) RNG ON

Figure 13: TVLA analysis of 1st-order `SecMatVec`. Mean trace and t-test results with RNG ON (1M traces). The ± 5.079 threshold is marked by red lines.



(a) Mean Trace



(b) RNG ON

Figure 14: TVLA analysis of 1st-order SecQuad. Mean trace and t-test results with RNG ON (1M traces). The ± 5.283 threshold is marked by red lines.