Lightweight Hardware Accelerator for the UOV Signature Scheme with Oil Space Blinding

Florian Krieger, Maciej Czuprynko and Sujoy Sinha Roy

Institute of Information Security, Graz University of Technology, Graz, Austria
firstname.lastname@tugraz.at

Abstract. In reaction to the emerging quantum threat, the National Institute of Standards and Technology (NIST) seeks post-quantum secure digital signature schemes. NIST's ongoing competition recently advanced to the third round, in which the Unbalanced Oil and Vinegar scheme (UOV) is a promising candidate due to UOV's conservative design, small signatures, and performant signing and verification. While these benefits make UOV attractive, the implementation aspects for compact hardware acceleration of UOV remain underexplored. Existing hardware accelerators optimize for performance and have large memory footprints. Yet, constrained devices require compact and low-memory hardware support. At the same time, lowering the memory consumption of UOV without sacrificing performance is challenging due to UOV's large public keys and intermediate results. In addition, low-end devices are prone to power side-channel attacks and benefit from lightweight countermeasures. In this work, we focus on the open challenges of compact UOV hardware acceleration. We design a low-memory UOV architecture with runtime-flexible support of NIST security levels 1 to 5. Our architecture uses a load-balanced 16-lane datapath supporting the critical matrix operations of UOV. In addition, we lower UOV's peak memory consumption using a slicing technique and enable on-the-fly data generation via streamlined data realignment. We also include a pipelined Gaussian Elimination unit and a streaming approach for the large public keys needed for signature verification. In addition to compact hardware acceleration, we defend against differential power analysis attacks through a lightweight blinding countermeasure relying on UOV's equivalent keys. Our blinding requires random non-singular matrices, which are not trivial to sample efficiently in hardware. We therefore assess existing sampling approaches and propose a novel technique based on the lower-upper decomposition of matrices. The resulting blinding solely re-uses existing datapath components and limits the runtime overhead to less than 30%. Compared to existing UOV accelerators, our efficient FPGA design reaches up to 5.1× lower signing and 3.2× lower verification latency, while reducing the memory consumption between 2.2× and 16×.

Keywords: Hardware Acceleration, Post-Quantum Signatures, UOV, MAYO, Blinding

1 Introduction

The advances in quantum computing threaten conventional cryptographic schemes that rely on discrete logarithm and integer factorization problems. In reaction to this threat, the National Institute of Standards and Technology (NIST) is standardizing post-quantum secure signature schemes. This effort has already led to standards for lattice-based [NDL+24, AAC+22] and hash-based signatures [NC24]. To further diversify the portfolio of signature schemes, NIST currently drives a second standardization competition and nominates nine third-round candidates [ABC+26]. The third-round candidates include constructions relying on hard isogeny-, lattice-, MPC-in-the-head-, and multivariate-based problems. Therein, multivariate (MV) signature schemes are attractive candidates for NIST due to their small

signature sizes and performant signing and verification operations [ABC+26]. In particular, the Unbalanced Oil and Vinegar scheme (UOV) by Beullens *et al.* [BCD+25] outperforms other MV schemes in signing latency and signature sizes while maintaining conservative security guarantees [WP26]. These advantages of UOV come with the drawback of larger public keys. Nevertheless, UOV attracts broad interest and motivates research on UOV's mathematical security [Ran26, FI26] and implementation efficiency [BCH+23, BCC+24].

The implementation efficiency of UOV is particularly important, especially when deploying UOV on resource-constrained devices such as IoT nodes or automotive controllers. Therefore, works such as [BCH+23] optimize the UOV scheme for microcontrollers, but no compact UOV hardware accelerator for resource-constrained devices has been published. At the same time, implementing UOV efficiently in low-memory hardware is not trivial due to the large intermediate data and large public keys. In existing UOV hardware [BCH+23], the on-chip memory ranges between 216 kB and 1.6 MB for NIST security levels 1 to 5 and dominates the overall area consumption [HSK+24]. Thus, UOV accelerators consume far more memory than state-of-the-art Dilithium hardware [GJCJ23, NKZ+26], underlining the need for memory-oriented implementation techniques. In addition, side-channel timing and power leakage of unprotected implementations allow adversaries to retrieve the secret signing key. Hence, the works in [KNK+25, CGZ26] propose masked microcontroller implementations of UOV, but first-order masked UOV has a substantial runtime overhead of up to 12× in software. This overhead calls for lightweight countermeasures allowing trade-offs between side-channel resistance and computational efficiency.

In this work, we address these research gaps and investigate efficient and low-memory design principles for UOV hardware accelerators. Compared to existing UOV hardware designs, our proposed architecture lowers the on-chip memory consumption by up to 16×, which allows us to support the largest Uov-V security level with just 99 kB of on-chip storage. At the same time, we reach competitive signing and verification performance across NIST security levels 1 to 5. We also consider physical security and integrate a lightweight blinding countermeasure relying on UOV's equivalent keys. Our specific contributions are:

- We present implementation techniques to reduce the memory consumption of UOV hardware accelerators. These techniques include a slicing approach to serialize memory-centric operations and an efficient method for on-the-fly data generation and rearrangement. We also optimize for signing and verification performance within our low-memory setting by implementing a load-balanced 16-lane datapath and a pipelined Gaussian Elimination unit. The resulting UOV co-processor for FPGAs supports runtime-flexible selection of Uov-Ip, Uov-III, and Uov-V parameter sets and only requires 22 BRAMs. This shows the effectiveness of our optimizations and allows up to 16× BRAM reduction compared to existing UOV accelerators [BCH+23]. We also reach up to 5.1× and 3.2× lower cycle counts for signing and verification.
- We investigate a lightweight blinding countermeasure against differential power analysis attacks. The blinding relies on UOV's equivalent keys and re-randomizes the secret oil space basis matrix $\bar{\mathbf{O}}$ for each signing operation. This uses a single additional matrix multiplication $\bar{\mathbf{O}} \cdot \mathbf{R}$ with a random non-singular matrix $\mathbf{R}$. The randomization makes first-order DPA attacks more difficult, while avoiding high area and runtime overheads. In addition, our blinding is transparent to the remaining signing steps and allows straightforward combination with other countermeasures.
- We discuss different approaches to sample random non-singular matrices $\mathbf{R}$, which is needed in our blinding and in the masking scheme presented in [CGZ26]. Yet, sampling random non-singular matrices is not trivial to implement efficiently in hardware, as it either follows a hardware-unfriendly iterative construction or slow rejection sampling. We therefore propose a new approach for sampling non-singular matrices relying on the lower-upper (LU) decomposition. Our LU approach for matrix

sampling is efficient for hardware as it only re-uses existing datapath components. Thus, the area overhead of the LU-based blinding countermeasure is marginal, while the runtime overhead is between 21% and 29% for different security levels.

2 Background

2.1 Notation

Let $[n] = \{0, 1, \ldots, n-1\}$ be the set of non-negative integers smaller than n and $\mathbb{F}_q$ the finite field with q elements. Also, let $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$. We denote vectors and matrices over $\mathbb{F}_q$ using bold lowercase $\mathbf{v} \in \mathbb{F}_q^n$ and bold uppercase letters $\mathbf{M} \in \mathbb{F}_q^{n \times m}$, respectively. We index the i-th element of a length-n vector $\mathbf{v} \in \mathbb{F}_q^n$ as $\mathbf{v}[i]$ for $i \in [n]$. The element at the i-th row and j-th column of a matrix $\mathbf{M} \in \mathbb{F}_q^{n \times m}$ is denoted as $m_{i,j} = \mathbf{M}[i, j]$ for $i \in [n]$ and $j \in [m]$. Furthermore, we use $\mathbf{M}[i, :]$ and $\mathbf{M}[:, j]$ to denote the i-th row vector and j-th column vector of $\mathbf{M}$, respectively. We denote a set of m matrices $\{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_{m-1}\}$ as $\{\mathbf{P}_i\}_{i \in [m]}$. We write the $m \times m$ identity matrix as $\mathbf{I}_m$ and the set of non-singular $\mathbb{F}_q^{k \times k}$ matrices as $\mathrm{GL}_k(\mathbb{F}_q)$. We use $a \xleftarrow{\$} S$ when a is drawn uniformly at random from a set S.

2.2 The UOV Signature Scheme

The Unbalanced Oil and Vinegar scheme (UOV) [BCD+25] is one of the nine round-3 candidates in NIST's competition for additional post-quantum signature schemes [ABC+26]. UOV relies on the hardness of solving systems of multivariate quadratic equations, which has attracted decades of cryptanalytic attention. Moreover, UOV follows a conservative construction by avoiding performance-oriented tweaks of the underlying problem. Besides the deep cryptanalytic understanding, the main benefit of UOV is its small signature sizes and performant verification. Yet, UOV has larger public keys than other round-3 candidates. This is inherent to MV schemes and motivates other MV schemes to reduce public key size via modified security assumptions [ABC+26]. For example, the closely related MAYO scheme [BCC+25] introduces a new *emulsification technique* to reduce public key size at the cost of larger signatures. Nevertheless, the underlying operations of UOV and MAYO are very similar, which allows extending implementation techniques to both schemes.

Technical Details of UOV. The UOV scheme as defined in [BCD+25] uses a public map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$, where $\mathcal{P}(\mathbf{x}) = (p_0(\mathbf{x}), \ldots, p_{m-1}(\mathbf{x}))$. The polynomials $p_i(\mathbf{x}) : \mathbb{F}_q^n \to \mathbb{F}_q$ are homogeneous multivariate quadratic polynomials, where each p_i can be expressed as

$$p_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}_i \mathbf{x} = \mathbf{x}^\top \begin{bmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0}_{m \times v} & \mathbf{P}_i^{(3)} \end{bmatrix} \mathbf{x}. \tag{1}$$

The matrices $\mathbf{P}_i \in \mathbb{F}_q^{n \times n}$, $\mathbf{P}_i^{(1)} \in \mathbb{F}_q^{v \times v}$, and $\mathbf{P}_i^{(3)} \in \mathbb{F}_q^{m \times m}$ are upper-triangular. The $\mathbf{P}_i^{(2)} \in \mathbb{F}_q^{v \times m}$ matrices are dense with scheme parameters n, m, q, and $v = n - m$.

The signature $\mathbf{s} \in \mathbb{F}_q^n$ of a message hash $\mathbf{t} \in \mathbb{F}_q^m$ is the preimage of $\mathbf{t}$ under $\mathcal{P}$. This means that a valid signature $\mathbf{s}$ of the message hash $\mathbf{t}$ satisfies $\mathcal{P}(\mathbf{s}) = \mathbf{t}$. For proper parameter choices of n, m, and q, it is hard to find a valid preimage $\mathbf{s}'$ given $\mathcal{P}$ and $\mathbf{t}$, unless $\mathcal{P}$'s trapdoor information is known. This trapdoor information is an m-dimensional secret linear subspace O (also called the *oil space*), in which $\mathcal{P}$ vanishes, i.e., $\forall \mathbf{o} \in O : \mathcal{P}(\mathbf{o}) = \mathbf{0}_m$. To compute a signature $\mathbf{s} = \mathbf{v} + \mathbf{o}$ using the UOV trapdoor, the signer randomly chooses a

Algorithm 1 Signing in UOV using the compact secret key csk [BCD+25]

UOV.Sign(csk, μ) ▷ $\mathsf{csk} = \mathsf{seed}_{\mathsf{sk}}$, message μ

1: $\mathsf{seed}_{\mathsf{pk}}, \mathbf{O} \leftarrow \mathsf{Shake}_{256}(\mathsf{csk})$ ▷ $\mathbf{O} \in \mathbb{F}_q^{v \times m}$

2: $\bar{\mathbf{O}} \leftarrow \begin{pmatrix} \mathbf{O} \\ \mathbf{I}_m \end{pmatrix}$ ▷ $\bar{\mathbf{O}} \in \mathbb{F}_q^{n \times m}$

3: $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}, \{\mathbf{P}_i^{(2)}\}_{i\in[m]} \leftarrow \mathsf{AES}_{128}\mathsf{CTR}(\mathsf{seed}_{\mathsf{pk}})$ ▷ $\mathbf{P}_i^{(1)} \in \mathbb{F}_q^{v\times v}$, $\mathbf{P}_i^{(2)} \in \mathbb{F}_q^{v\times m}$

4: $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{\mathsf{salt\_len}}, \quad \mathbf{t} \leftarrow \mathsf{Shake}_{256}(\mu \,\|\, \mathsf{salt})$ ▷ $\mathbf{t} \in \mathbb{F}_q^m$

5: **for** $\mathsf{ctr} = 0$ **upto** 255 **do** ▷ Up to 256 signing attempts

6: $\quad \mathbf{v} \leftarrow \mathsf{Shake}_{256}(\mu \,\|\, \mathsf{salt} \,\|\, \mathsf{seed}_{\mathsf{sk}} \,\|\, \mathsf{ctr})$ ▷ $\mathbf{v} \in \mathbb{F}_q^v$

7: $\quad \mathbf{A} \leftarrow \mathbf{0}_{m\times m}, \quad \mathbf{y} \leftarrow \mathbf{0}_m$ ▷ Initialize $\mathbf{A}$ and $\mathbf{y}$ with zero

8: $\quad$ **for** $i = 0$ **upto** $m-1$ **do**

9: $\quad\quad \mathbf{S}_i \leftarrow \left(\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)^\top} \;\middle|\; \mathbf{P}_i^{(2)} \right) \cdot \bar{\mathbf{O}} = (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)^\top}) \cdot \mathbf{O} + \mathbf{P}_i^{(2)}$ ▷ $\mathbf{S}_i \in \mathbb{F}_q^{v\times m}$

10: $\quad\quad \mathbf{A}[i,:] \leftarrow \mathbf{v}^\top \mathbf{S}_i$

11: $\quad\quad \mathbf{y}[i] \leftarrow \mathbf{v}^\top \mathbf{P}_i^{(1)} \mathbf{v}$

12: $\quad$ **if** $\mathbf{A}$ is invertible **then**

13: $\quad\quad$ Solve $\mathbf{A}\mathbf{x} = \mathbf{t} - \mathbf{y}$ for $\mathbf{x}$

14: $\quad\quad \mathbf{s} \leftarrow \begin{pmatrix} \mathbf{v} \\ \mathbf{0}_m \end{pmatrix} + \bar{\mathbf{O}} \cdot \mathbf{x}$ ▷ $\mathbf{s} \in \mathbb{F}_q^n$

15: $\quad\quad$ **return** $\sigma = (\mathbf{s}, \mathsf{salt})$

16: **return** $\perp$

Algorithm 2 Verification in UOV using the compact public key cpk [BCD+25]

UOV.Verify$(\mathsf{cpk}, \mu, \sigma)$ ▷ $\mathsf{cpk} = (\mathsf{seed}_{\mathsf{pk}}, \{\mathbf{P}_i^{(3)}\}_{i\in[m]})$, message μ, signature $\sigma = (\mathbf{s}, \mathsf{salt})$

1: $\mathbf{t} \leftarrow \mathsf{Shake}_{256}(\mu \,\|\, \mathsf{salt}), \quad \mathbf{t}' \leftarrow \mathbf{0}_n$ ▷ $\mathbf{t}, \mathbf{t}' \in \mathbb{F}_q^m$

2: $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}, \{\mathbf{P}_i^{(2)}\}_{i\in[m]} \leftarrow \mathsf{AES}_{128}\mathsf{CTR}(\mathsf{seed}_{\mathsf{pk}})$ ▷ $\mathbf{P}_i^{(1)} \in \mathbb{F}_q^{v\times v}$, $\mathbf{P}_i^{(2)} \in \mathbb{F}_q^{v\times m}$

3: **for** $i = 0$ **upto** $m-1$ **do** $\mathbf{t}'[i] \leftarrow \mathbf{s}^\top \begin{bmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0}_{m\times v} & \mathbf{P}_i^{(3)} \end{bmatrix} \mathbf{s}$

4: **return** $\mathbf{t} == \mathbf{t}'$

vinegar vector $\mathbf{v}$ and computes the *oil vector* $\mathbf{o} \in O$ via solving the linear equation system

$$\mathcal{P}(\mathbf{s}) = \mathcal{P}(\mathbf{v}+\mathbf{o}) = \underbrace{\mathcal{P}(\mathbf{v})}_{\text{const.}} + \underbrace{\mathcal{P}(\mathbf{o})}_{=\mathbf{0}_m} + \underbrace{\mathcal{P}'(\mathbf{v},\mathbf{o})}_{\text{linear in } \mathbf{o}} = \mathbf{t}$$

for $\mathbf{o}$, where $\mathcal{P}'(\mathbf{x},\mathbf{y}) = \mathcal{P}(\mathbf{x}+\mathbf{y}) - \mathcal{P}(\mathbf{x}) - \mathcal{P}(\mathbf{y})$ is the differential of $\mathcal{P}$ [BCD+25, BCC+25].

UOV Algorithms. Building upon the described UOV trapdoor, the signature scheme consists of three algorithms: key generation, signing, and verification. In the NIST submission, the authors of UOV propose compact key (csk, cpk) and expanded key (esk, epk) variants. In the compact key variant, csk and cpk contain the minimal data required to regenerate the key material via seed expansion. The expanded keys esk and epk contain additional precomputed data to speed up signing and verification. Since we target a low-memory hardware design, we focus on the compact key variant of UOV, as it reduces key sizes and peak memory consumption.

The compact key generation of UOV [BCD+25, Figure 2] first samples a random $\mathsf{seed}_{\mathsf{sk}}$, which generates the $\mathsf{seed}_{\mathsf{pk}}$ and $\mathbf{O} \in \mathbb{F}_q^{v\times m}$ using Shake_{256}. Therein, $\mathbf{O}$ forms the dense upper block of the basis matrix $\bar{\mathbf{O}} = \left(\mathbf{O}^\top \;\middle|\; \mathbf{I}_m \right)^\top \in \mathbb{F}_q^{n\times m}$, whose columns span the secret oil

Table 1: Round-2 parameter sets and key sizes of UOV. All sizes in bytes. [BCD+25]

Param. Set	Sec. Lvl.	n	m	v	q	$\lvert\mathsf{epk}\rvert$	$\lvert\mathsf{esk}\rvert$	$\lvert\mathsf{cpk}\rvert$	$\lvert\mathsf{csk}\rvert$	$\lvert\sigma\rvert$
Uov-Ip	1	112	44	68	256	278,432	237,896	43,576	32	128
Uov-Is	1	160	64	96	16	412,160	348,704	66,576	32	96
Uov-III	3	184	72	112	256	1,225,440	1,044,320	189,232	32	200
Uov-V	5	244	96	148	256	2,869,440	2,436,704	446,992	32	260

space O. Next, the upper-triangular $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}$ and the dense $\{\mathbf{P}_i^{(2)}\}_{i\in[m]}$ matrices are sampled from AES in counter mode $\mathsf{AES}_{128}\mathsf{CTR}(\mathsf{seed}_{\mathsf{pk}})$. Finally, the $\{\mathbf{P}_i^{(3)}\}_{i\in[m]}$ matrices are computed such that $\mathcal{P}$ vanishes over O. The output of compact key generation is the compact secret key $\mathsf{csk} = \mathsf{seed}_{\mathsf{sk}}$ and the compact public key $\mathsf{cpk} = (\mathsf{seed}_{\mathsf{pk}}, \{\mathbf{P}_i^{(3)}\}_{i\in[m]})$.

Algorithm 1 presents UOV's signing procedure using the compact secret key csk. Signing first expands $\mathsf{seed}_{\mathsf{sk}}$ to obtain $\mathsf{seed}_{\mathsf{pk}}$ and $\mathbf{O} \in \mathbb{F}_q^{v\times m}$. Subsequently, $\bar{\mathbf{O}}$ is constructed by combining $\mathbf{O}$ with $\mathbf{I}_m$, and public matrices $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}, \{\mathbf{P}_i^{(2)}\}_{i\in[m]}$ are sampled via $\mathsf{AES}_{128}\mathsf{CTR}$ in line 3. Then, a random salt is sampled and hashed together with the message to yield the target vector $\mathbf{t}$. In the following loop, UOV samples a random vinegar vector $\mathbf{v}$ and constructs a system of linear equations $\mathbf{Ax} = \mathbf{t} - \mathbf{y}$. When the system is not solvable, UOV samples a new vinegar vector and repeats the procedure. Otherwise, the signature is computed as in lines 13 - 14. Note that the operation in line 9 can leverage the identity matrix contained in $\bar{\mathbf{O}}$, which simplifies the computation to $\mathbf{S}_i \leftarrow (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)^\top}) \cdot \mathbf{O} + \mathbf{P}_i^{(2)}$.

The verification of UOV is shown in Algorithm 2. Verification expands the compact public key cpk to the full $\{\mathbf{P}_i\}_{i\in[m]}$ matrices, and matches the hashed vector $\mathbf{t}$ with the evaluation of the public map $\mathcal{P}(\mathbf{s})$. The signature is accepted if and only if $\mathcal{P}(\mathbf{s}) == \mathbf{t}$.

UOV Parameter Sets. The round-2 submission of UOV specifies four parameter sets, as shown in Table 1. The Uov-Ip and Uov-Is parameter sets satisfy NIST's security level 1 and optimize for smaller public keys and smaller signatures, respectively. Uov-III and Uov-V correspond to NIST's security levels 3 and 5. Note that Uov-Is uses $\mathbb{F}_{16}$, which differs from the field $\mathbb{F}_{256}$ used in the other parameter sets. Next to the security level dependent parameters from Table 1, UOV specifies $\mathsf{sk\_seed\_len} = 256$, $\mathsf{pk\_seed\_len} = 128$, and $\mathsf{salt\_len} = 128$, independent of the security level. Since no round-3 parameters have been published, we use the up-to-date round-2 parameter sets for our implementation and emphasize that our contributions can be extended to future parameter updates.

2.3 Gaussian Elimination

The Gaussian Elimination (GE) is a well-known algorithm to solve linear equation systems, as required in UOV's signing (line 13 in Algorithm 1). To solve a system $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{x}$, GE first computes the Echelon Form of the augmented system matrix $\left(\begin{array}{c|c} \mathbf{A} & \mathbf{b} \end{array}\right)$. Then, Back Substitution computes the result vector $\mathbf{x}$, as shown in Algorithm 3. Since GE is a central component in different PQC schemes, it has been a target for hardware acceleration in [CNU26, FG18, BCH+23]. These works often use systolic arrays for GE, which increases area consumption due to numerous processing elements. In addition, the GE in UOV operates over secret data and therefore requires a constant-time implementation [BCC+24].

3 Low-Area Hardware Architecture for UOV

In this section, we present our low-memory UOV hardware architecture for FPGAs, which supports signing and verification for runtime-configurable Uov-Ip, Uov-III, and Uov-V parameter sets. The key challenge of a low-memory UOV implementation is the handling

Algorithm 3 Gaussian Elimination for system solving

GE($\mathbf{A} \in \mathbb{F}_q^{m \times m}, \mathbf{b} \in \mathbb{F}_q^m$) ▷ Returns $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{b}$ or $\perp$
1: **for** $k = 0$ **upto** $m - 1$ **do** ▷ Step 1: Compute Echelon Form
2: **for** $i = k + 1$ upto $m - 1$ **do** ▷ Pivot search, must be constant time
3: **if** $\mathbf{A}[k, k] = 0$ and $\mathbf{A}[i, k] \neq 0$ **then**
4: $\mathbf{A}[k, :] \leftarrow \mathbf{A}[k, :] + \mathbf{A}[i, :], \quad \mathbf{b}[k] \leftarrow \mathbf{b}[k] + \mathbf{b}[i]$
5: **if** $A[k, k] = 0$ **then return** $\perp$ ▷ $\mathbf{A}$ is singular
6: $p \leftarrow \mathbf{A}[k, k]^{-1}$ ▷ Pivot's inverse, also denoted as `piv_inv`
7: $\mathbf{A}[k, :] \leftarrow p \cdot \mathbf{A}[k, :], \quad \mathbf{b}[k] \leftarrow p \cdot \mathbf{b}[k]$ ▷ Scale pivot row
8: **for** $i = k + 1$ **upto** $m - 1$ **do** ▷ Eliminate entries below the pivot
9: $e \leftarrow \mathbf{A}[i, k]$
10: $\mathbf{A}[i, :] \leftarrow \mathbf{A}[i, :] - e \cdot \mathbf{A}[k, :], \quad \mathbf{b}[i] \leftarrow \mathbf{b}[i] - e \cdot \mathbf{b}[k]$
11: **for** $k = m - 1$ **downto** 0 **do** ▷ Step 2: Compute Back Substitution
12: $\mathbf{x}[k] \leftarrow \mathbf{b}[k] - \sum_{j=k+1}^{m-1} \mathbf{A}[k, j] \cdot \mathbf{x}[j]$
13: **return** $\mathbf{x} \in \mathbb{F}_q^m$

of large matrices. This is reflected in the high on-chip memory consumption of existing MV acceleration works [BCH+23, HSK+24, SMA+24], requiring between 205 kB and 1.6 MB of on-chip memory for security levels 1 to 5. Instead, our architecture only needs 99 kB for supporting up to Uov-V while maintaining competitive performance. We reach this memory efficiency via a slicing approach to serialize memory-centric operations and an on-the-fly sampling of $\{\mathbf{P}_i\}_{i \in [m]}$. We also optimize GE by reducing memory port conflicts. These design techniques and their application to UOV are presented in the following.

3.1 Overall UOV Architecture

Figure 1 shows our overall UOV architecture. The I/O module (bottom right in Figure 1) interfaces between the accelerator and the host CPU. The I/O module allows the host CPU to access the BRAMs and offers memory-mapped registers for $\mathsf{seed}_{\mathsf{sk}}$, $\mathsf{seed}_{\mathsf{pk}}$, message length, and other control signals. Moreover, the I/O module selects security level, signing and verification operations, and configures the Overall Controller and the Gaussian Elimination Controller. The processing units – consisting of Datapath, Field Inversion, Shake_{256}, and $\mathsf{AES}_{128}\mathsf{CTR}$ – connect to the memory subsystem via the central Data Bus. The memory subsystem includes five BRAM banks, with a word size of 128 bits. This word size aligns with the $\mathsf{AES}_{128}\mathsf{CTR}$ output and the datapath bitwidth, as we will explain later. Note that BRAM_A and BRAM_vs offer dual read ports for the high reading throughput required in computing the GE and the quadratic polynomials (i.e., $\mathbf{v}^\top \mathbf{P}_i^{(1)} \mathbf{v}$ and $\mathbf{s}^\top \mathbf{P}_i \mathbf{s}$). In the next subsections, we detail the individual components of our UOV architecture.

3.2 Datapath Architecture and Slicing

The most time-consuming operations in UOV's signing and verification are matrix-matrix and matrix-vector multiplications. Signing spends the major latency share on setting up the linear equation system $\mathbf{Ax} = \mathbf{t} - \mathbf{y}$ (lines 8 - 11 in Algorithm 1). Verification mostly consists of the computation of the quadratic polynomials $\mathbf{s}^\top \mathbf{P}_i \mathbf{s}$ (line 3 in Algorithm 2). In addition to their high latency, storing these operations' large intermediate results is not feasible in a low-memory design since the $\{\mathbf{P}_i\}_{i \in [m]}$ and $\{\mathbf{S}_i\}_{i \in [m]}$ matrices consume between 400 kB (Uov-Ip) and 4 MB (Uov-V). These storage requirements exceed the BRAM capacity of low-end FPGAs [AMD21]. We therefore optimize the critical matrix operations and introduce a *slicing* approach to efficiently support multiple parameter sets

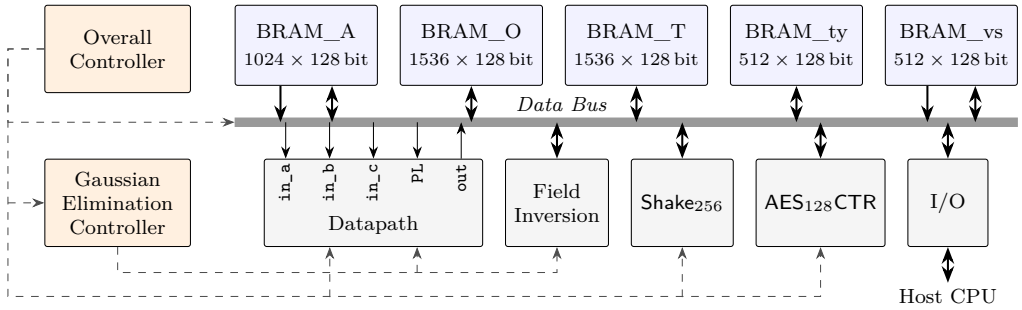


Figure 1: Overall UOV architecture with memory depths to support up to Uov-V.

and to lower the memory pressure in signing and verification.

Our slicing approach uses the observation that all critical matrix operations in UOV follow the pattern shown in Figure 2. In both signing and verification, the shown operation multiplies a set of matrices $\{\mathbf{C}_i\}_{i\in[m]}$ by a single operand from the left ($\mathbf{a}^\top$) or from the right ($\mathbf{B}$ or $\mathbf{b}$)[1]. Note that there are always m matrices in the set $\{\mathbf{C}_i\}_{i\in[m]}$ and the operation in Figure 2 results in m vectors or scalars.

One extreme case to implement this operation is to fully unroll m parallel processing lanes, where each lane computes over one $\mathbf{C}_i$ matrix. However, this would lead to a high area consumption and cannot efficiently support multiple parameter sets. To support multiple UOV parameter sets with m between 44 and 96, fully unrolling the matrix operations would need 96 processing lanes. Yet, when computing smaller parameter sets with $m < 96$, several processing lanes would remain underutilized. Furthermore, fully unrolling the operation requires storing all intermediate results at once, which severely increases memory consumption. Another approach is to just instantiate one processing lane, which would reduce area consumption at the cost of high latency. Instead, we balance performance, area, and memory consumption by fine-tuning the number of processing lanes. In our architecture, we instantiate a datapath with 16 processing lanes computing over a subset of 16 $\mathbf{C}_i$ matrices at a time. Choosing 16 lanes is an efficient trade-off since it aligns with the bitwidth of $\mathsf{AES}_{128}\mathsf{CTR}$ and our memory subsystem.

We call each set of 16 $\mathbf{C}_i$ matrices a *slice* (indicated by matching colors in Figure 2) and iterate over the $s = \lceil m/16 \rceil$ slices sequentially. This slicing approach has not been reported in existing works and comes with two benefits. First, it allows us to only instantiate BRAM memory for the intermediate results of one slice and reuse the same memory across consecutive slice computations. Second, slicing enables us to support Uov-Ip, Uov-III, and Uov-V parameter sets in a unified architecture with low memory overhead. Depending on the security level, we iterate over $s = 3$, 5, or 6 slices and perform the same operation sequence within each slice. Note that slicing can also be applied to Uov-Is, which uses $q = 16$. However, due to the different finite field, we do not support Uov-Is in our architecture and leave its implementation for future engineering effort.

Datapath Architecture. As already mentioned, our datapath operates over a single slice with its 16 $\mathbf{C}_i$ matrices. Figure 3a shows our 16-lane datapath performing parallel multiply and accumulate operations over the `in_c`[] operands (mapping to $\mathbf{C}_i$ in Figure 2). The scalar inputs `in_a` and `in_b` map to $\mathbf{a}$ and $\mathbf{B}$ in Figure 2 and compute $\texttt{out}[i] = \texttt{in\_a}\cdot\texttt{in\_b}\cdot\texttt{in\_c}[i]$ for $i \in [16]$. In addition, the `preload`[] input allows initializing the MAC registers, thereby supporting vector additions like in Algorithm 1, line 14, where $\mathbf{v}$ is preloaded into the registers and the result of $\bar{\mathbf{O}}\mathbf{x}$ is accumulated to $\mathbf{v}$. With this datapath, we can efficiently

[1]We use generic symbols $\{\mathbf{C}_i\}_{i\in[m]}$, $\mathbf{a}$, $\mathbf{B}$, and $\mathbf{b}$ to refer to different operands in UOV's signing and verification. The symbols match the datapath port names `in_a`, `in_b`, and `in_c`, which we introduce later.

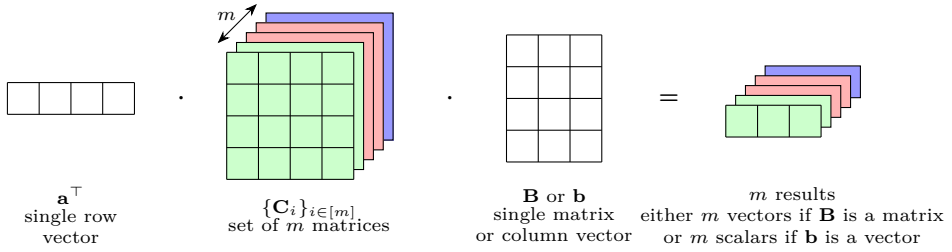


Figure 2: Illustration of our slicing approach. Matrices of the same color are processed in parallel. This shows two $\mathbf{C}_i$ matrices per slice for demonstration purposes, while our datapath processes 16 $\mathbf{C}_i$ matrices per slice.

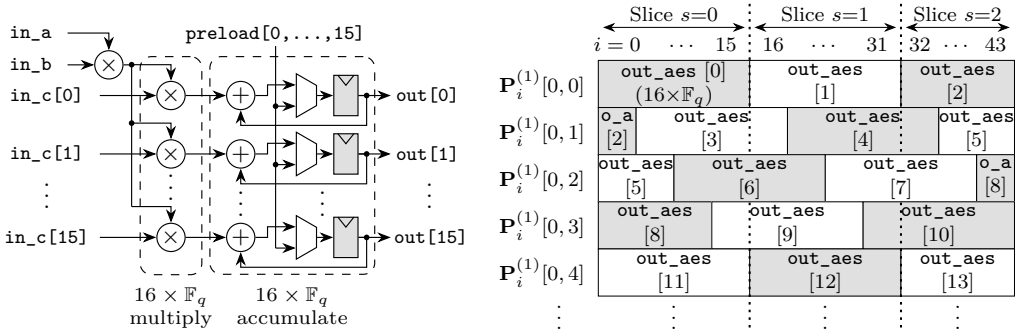
compute the critical sliced matrix operations in signing and verification. When performing simple, non-sliced matrix operations such as in line 14 of Algorithm 1, the 16 multipliers are used to compute 16 individual elements of the resulting matrix. This lightweight computation offers an efficient trade-off between area and performance.

3.3 On-the-fly Data Sampling

To reduce the memory consumption in signing and verification, we extend the slicing approach by an on-the-fly sampling of the $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in[m]}$ matrices. Specifically, we use 10-round $\mathsf{AES}_{128}\mathsf{CTR}$ to generate the matrix elements of one slice and directly forward the sampled data to the `in_c` port of the datapath. Since the AES-based data sampling is critical for performance, we use fully unrolled and pipelined $\mathsf{AES}_{128}\mathsf{CTR}$, generating 16 field elements (128 bits) per clock cycle.

However, combining the slicing approach and the on-the-fly generation of $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}$ and $\{\mathbf{P}_i^{(2)}\}_{i\in[m]}$ is not trivial due to UOV's specified data layout. The problem lies in the fact that $\mathsf{AES}_{128}\mathsf{CTR}$ outputs 16 field elements at once, and these 16 field elements belong to 16 different $\mathbf{P}_i$ matrices. At the same time, 16 does not divide m for Uov-Ip and Uov-III. Therefore, a single $\mathsf{AES}_{128}\mathsf{CTR}$ output can cross the slice boundaries and overflow into subsequent slices. To illustrate this problem, consider Uov-Ip with $m = 44 = 2\cdot 16+12$, as shown in Figure 3b. Therein, the first $\mathsf{AES}_{128}\mathsf{CTR}$ output (denoted as `aes_out`[0]) returns 16 field elements $\mathbf{P}_0^{(1)}[0,0]$ to $\mathbf{P}_{15}^{(1)}[0,0]$. All these elements belong to slice number $s = 0$, and the computation of slice $s = 0$ can directly start in this case. Yet, `aes_out`[2] generates the elements $\mathbf{P}_{32}^{(1)}[0,0]$ to $\mathbf{P}_{43}^{(1)}[0,0]$ and $\mathbf{P}_0^{(1)}[0,1]$ to $\mathbf{P}_3^{(1)}[0,1]$. While $\mathbf{P}_{32}^{(1)}[0,0]$ to $\mathbf{P}_{43}^{(1)}[0,0]$ belong to slice number $s = 2$, $\mathbf{P}_0^{(1)}[0,1]$ to $\mathbf{P}_3^{(1)}[0,1]$ are elements from slice $s = 0$ (see Figure 3b). Since `aes_out`[2] only returns the first four elements of slice 0, another call to `aes_out`[3] is required to obtain all required elements for the $s = 0$ computation. This misaligned seed expansion complicates hardware design. Existing works [HSK+24, BCH+23] resolve this problem through memory-based data alignment, but at the cost of high BRAM consumption.

We propose two solutions to combine on-the-fly seed expansion and a streamlined data realignment. The first solution is to only instantiate one $\mathsf{AES}_{128}\mathsf{CTR}$ core. The single $\mathsf{AES}_{128}\mathsf{CTR}$ core computes both outputs contributing to one slice in two consecutive clock cycles (e.g., `aes_out`[2] and `aes_out`[3] to obtain all $\mathbf{P}_i^{(1)}[0,1]$, $i \in [16]$ for $s = 0$, as in Figure 3b). Then, the two outputs are blended via shifts and conditional register updates to align with the slice boundary. This blending operation is lightweight since each output byte position can only originate from at most $\max\{16/\gcd(16,m) : m \in \{44,72,96\}\} = 4$ AES positions. Thus, blending consists of a 4:1 multiplexing operation. The described first approach returns one 16-lane input to the datapath every two cycles but only uses



(a) Datapath. (b) AES layout (`o_a` = `out_aes`).

Figure 3: (a) 16-lane multiply-and-accumulate datapath to compute within a slice. (b) $\mathsf{AES}_{128}\mathsf{CTR}$ output data layout for UOV-Ip ($m = 44$).

one $\mathsf{AES}_{128}\mathsf{CTR}$ unit. Alternatively, our second approach instantiates two $\mathsf{AES}_{128}\mathsf{CTR}$ cores, where one computes `aes_out`$[i]$ and one computes `aes_out`$[i+1]$ in parallel. The two outputs are then blended in a pipelined fashion to form the 16-lane datapath input. Although the second $\mathsf{AES}_{128}\mathsf{CTR}$ core increases the area footprint by roughly 3k lookup tables (which is 9% of the whole LUT consumption), we opt for the second approach to reduce signing and verification latency by up to 2×. Another benefit of on-the-fly sampling is the parallelization of $\mathsf{AES}_{128}\mathsf{CTR}$ computations and arithmetic operations within a single pipeline. This reduces latency compared to serialized $\mathsf{AES}_{128}\mathsf{CTR}$ and matrix operations.

3.4 Gaussian Elimination

Another critical step in the signing operation is solving the linear equation system $\mathbf{Ax} = \mathbf{t} - \mathbf{y}$. The GE, as shown in Algorithm 3, is a common choice for this step. While many existing implementations of GE use resource-intensive systolic arrays, we aim to reutilize our datapath from Section 3.2 to reduce area consumption. Thus, we choose an iterative approach to compute the GE, where we load data from BRAM to our datapath, perform batched operations over up to 16 lanes, and store the result back to BRAM. However, fully pipelining our load-compute-store approach for GE requires reading up to two operands and storing one result back to BRAM per clock cycle (see lines 4, 10, and 12 in Algorithm 3). This causes BRAM port conflicts since the FPGA's BRAM banks only support two independent accesses in each cycle, one per port. In our architecture, we resolve the port conflict by using BRAM_A, which stores the system matrix $\mathbf{A}$, BRAM_ty storing the vector $\mathbf{y}' = \mathbf{t} - \mathbf{y}$, and BRAM_T, which mirrors the content of BRAM_A and BRAM_ty. In particular, BRAM_T holds copies of $\mathbf{A}$ and $\mathbf{y}'$, and every write to $\mathbf{A}$ and $\mathbf{y}'$ also updates BRAM_T along with BRAM_A and BRAM_ty. This approach offers us two read ports and one write port, which can operate independently. The resulting pipelined GE lowers our signing latency. At the same time, we reuse BRAM_T, which is already part of our design, thereby avoiding additional memory consumption.

Yet, pipeline hazards remain and require careful consideration. These hazards occur in the late stages of the Echelon Form and Back Substitution steps, where the production-to-consumption window of intermediate data is smaller than the pipeline latency. We avoid these hazards by monitoring the data dependencies in our control logic. This happens in a data-independent way by counting the cycles between consecutive operations involving the same data. If the counter is below the pipeline latency threshold, we wait the minimal cycle count required to commit the previous data value, thereby improving the performance.

Our constant-time implementation of GE takes between 18% and 30% of the overall

signing latency. In addition, GE effectively reuses the existing BRAMs and the datapath by parallelizing the row operations in the 16 datapath lanes. Finally, control registers allow for flexibly changing the UOV parameter set at runtime.

3.5 Verification Support

The verification in UOV, as explained in Algorithm 2, computes the target hash $\mathbf{t}$ via Shake_{256} and matches $\mathbf{t}$ with the result of $\mathbf{s}^\top \mathbf{P}_i \mathbf{s}$. Since computing $\mathbf{s}^\top \mathbf{P}_i \mathbf{s}$ is very similar to computing $\mathbf{v}^\top \mathbf{P}_i^{(1)} \mathbf{v}$ in signing, we do not need additional datapath components to support verification. However, verification involves the $\{\mathbf{P}_i^{(3)}\}_{i\in[m]}$ matrices, which are part of the cpk. The $\{\mathbf{P}_i^{(3)}\}_{i\in[m]}$ matrices are large and consume up to 437 kB for UOV-V, which cannot be stored in our architecture.

Instead of increasing our memory consumption to support verification, we propose a streaming approach for verification. Our streaming approach operates on top of slicing and computes each slice of $\mathbf{s}^\top \mathbf{P}_i \mathbf{s}$ in two steps. In the first step, we only compute $t_i = \mathbf{s}[:v]^\top \left[\mathbf{P}_i^{(1)} \mid \mathbf{P}_i^{(2)}\right] \mathbf{s}[:v]$ using on-the-fly data sampling of $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(2)}$ in a sliced manner. Then, we compute the remaining sum $\mathbf{s}[v:]^\top \mathbf{P}_i^{(3)} \mathbf{s}[v:] = \sum_{j=0}^{m-1} \sum_{k=j}^{m-1} \mathbf{s}[v+j]\mathbf{s}[v+k]\mathbf{P}_i^{(3)}[j,k]$ and accumulate it to the previous result t_i. We note that each element $\mathbf{P}_i^{(3)}[j,k]$ is only used once for computing the quadratic polynomials. Thus, the elements of $\mathbf{P}_i^{(3)}$ can be loaded from off-chip memory in a streaming fashion. As soon as a data slice of $\mathbf{P}_i^{(3)}$ arrives at the co-processor, it is directly consumed to compute $\mathbf{s}[v:]^\top \mathbf{P}_i^{(3)} \mathbf{s}[v:]$. Therefore, we avoid storing $\mathbf{P}_i^{(3)}$ on chip.

4 Lightweight Blinding Countermeasure

Cryptographic implementations running on attacker-accessible devices demand countermeasures to make side-channel attacks impractical. One powerful countermeasure against differential power analysis attacks [KJJ99] is masking, which splits secret variables into $d > 1$ randomized shares. This randomization breaks the correlation between the device's power consumption and the secret, as long as the attacker cannot observe all d shares simultaneously. The strong security guarantees of masking, however, come at the price of high overheads in terms of runtime, memory consumption, and randomness demands. Considering the UOV scheme, two masking approaches have been proposed for software platforms [KNK+25, CGZ26]. The work in [KNK+25] presents an end-to-end masked UOV implementation with a 12× runtime overhead for first-order masking. The authors of [CGZ26] present a different masking method for solving linear systems at the cost of up to 7.1× increased signing runtime. Although these software overheads do not directly translate to hardware, masking still causes increased latency, area, and memory consumption.

A more lightweight countermeasure is blinding [Koc96]. Blinding randomizes internal variables without splitting them into shares and thus hides the secret in the power trace. It is important to note that blinding has weaker security guarantees than masking, but it causes less overhead and thus suits resource-constrained devices. Moreover, it is straightforward to combine blinding with other countermeasures to further increase physical security. This motivates us to investigate blinding for UOV as a case study for MV schemes.

4.1 Threat Model

Differential power analysis (DPA) [KJJ99] is a statistical side-channel attack, where the attacker collects power traces of many executions of the algorithm, where each execution

uses the same static secret key. Since the power consumption correlates with secret key-dependent intermediate values, the attacker can match the observed power consumption to a key hypothesis. One method to complicate DPA is to randomize the secret such that each execution of the algorithm has randomized power leakage. This randomization can be achieved by our blinding approach, which turns the long-term static secret key into randomized ephemeral keys to make DPA more difficult. We also mitigate timing side-channel attacks with a constant-time implementation. Yet, we consider higher-order DPA attacks outside of our threat model. In addition, we do not consider single-trace attacks or fault attacks, which have been explored in prior work [SMA+24, ACK25, ACK+23, BDSK25, JD26]. Nevertheless, our blinding countermeasure is transparent to the remaining operations in signing. This allows naturally combining blinding with other countermeasures such as masking or shuffling. Therefore, the lightweight blinding is not a replacement for masking, but it can serve as an additional measure to complicate side-channel attacks.

4.2 Blinding UOV using Equivalent Keys

It is well known that MV schemes do not have a unique signing key [WP05]. Instead, MV schemes have a large class of equivalent signing keys, where each equivalent key generates a valid signature under the public key. This allows randomly choosing a new equivalent key in each signing operation, leading to a randomized (blinded) signature computation.

Previous work [PSN22] has used this observation for blinding the Rainbow [DS05] MV scheme from NIST's first PQC competition. However, the security of Rainbow has been broken [Beu22] and the blinding approach for Rainbow does not directly extend to UOV. This is due to the structure of Rainbow's public map $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$, composed from the secret central map $\mathcal{F}$, and two invertible affine maps $\mathcal{S}$ and $\mathcal{T}$. The authors of [PSN22] blind the secret maps using two random linear maps $\mathcal{A}$ and $\mathcal{B}$ such that

$$\mathcal{P} = (\mathcal{S} \circ \mathcal{A}^{-1}) \circ (\mathcal{A} \circ \mathcal{F} \circ \mathcal{B}) \circ (\mathcal{B}^{-1} \circ \mathcal{T}).$$

As presented in Section 2.2, UOV differs from Rainbow and does not explicitly use the individual secret maps, preventing a direct application of [PSN22]. Instead, UOV uses the basis matrix $\bar{\mathbf{O}} \in \mathbb{F}_q^{n \times m}$ whose column space defines the secret oil space O. Yet, the basis is not unique, and there exist numerous basis matrices defining the same oil space O. We therefore propose to blind the basis matrix by computing

$$\bar{\mathbf{O}}_b = \bar{\mathbf{O}} \cdot \mathbf{R}$$

with a random, non-singular $\mathbf{R} \in \mathbb{F}_q^{m \times m}$, as shown in Algorithm 4. This yields an equivalent, blinded basis matrix $\bar{\mathbf{O}}_b$ spanning the same O. The subsequent signing steps in lines 8 - 14 of Algorithm 4 use $\bar{\mathbf{O}}_b$ to compute the linear system $\mathbf{Ax} = \mathbf{t} - \mathbf{y}$ and solve for $\mathbf{x}$. Only at the end of the signature computation (line 14), the multiplication of $\bar{\mathbf{O}}_b \cdot \mathbf{x}$ cancels the effect of blinding, and the resulting signature $\mathbf{s}$ is *identical* to the unblinded case. Note that blinding does not affect the rank of $\mathbf{A}$ and, hence, the solvability of $\mathbf{Ax} = \mathbf{t} - \mathbf{y}$.

Algorithm 4 shows our blinded UOV signing procedure with the changes to the original UOV signing highlighted in blue. During key generation, we store the expanded and blinded basis matrix $\bar{\mathbf{O}}_b = \bar{\mathbf{O}} \cdot \mathbf{R}$ in persistent memory. Then, signing loads the blinded basis matrix $\bar{\mathbf{O}}_b$ from persistent storage (line 3). In each signing attempt, we re-randomize the basis matrix with a randomly sampled non-singular matrix $\mathbf{R}$ (lines 6 - 7). After a successful signing attempt, we store the updated $\bar{\mathbf{O}}_b$ matrix again in persistent memory (line 15) for its use in future signing operations. The blinding also scrambles the structure of $\bar{\mathbf{O}}_b$, which no longer has $\mathbf{I}_m$ in the lower block. Therefore, the operation in line 10 cannot leverage the optimized computation as in Algorithm 1, line 9.

Since $\bar{\mathbf{O}}$ without blinding is a static secret involved in numerous suboperations, it is an attractive attack target. With our blinding approach, the secret basis matrix $\bar{\mathbf{O}}_b$ is

Algorithm 4 Blinded signing in UOV

UOV.BlindedSign$(\mathsf{seed_{pk}}, \mu)$ ▷ message μ

1: $\{\mathbf{P}_i^{(1)}\}_{i\in[m]}, \{\mathbf{P}_i^{(2)}\}_{i\in[m]} \leftarrow \mathsf{AES_{128}CTR}(\mathsf{seed_{pk}})$ ▷ $\mathbf{P}_i^{(1)} \in \mathbb{F}_q^{v\times v}$, $\mathbf{P}_i^{(2)} \in \mathbb{F}_q^{v\times m}$

2: $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{\mathsf{salt\_len}}$, $\mathbf{t} \leftarrow \mathsf{Shake}_{256}(\mu \,\|\, \mathsf{salt})$ ▷ $\mathbf{t} \in \mathbb{F}_q^m$

3: Load $\bar{\mathbf{O}}_b$ from persistent storage.

4: **for** $\mathsf{ctr} = 0$ **upto** 255 **do**

5: $\quad \mathbf{v} \leftarrow \mathsf{Shake}_{256}(\mu \,\|\, \mathsf{salt} \,\|\, \mathsf{seed_{sk}} \,\|\, \mathsf{ctr})$

6: $\quad \mathbf{R} \xleftarrow{\$} \mathrm{GL}_m(\mathbb{F}_q)$ ▷ Sample fresh blinding matrix

7: $\quad \bar{\mathbf{O}}_b \leftarrow \bar{\mathbf{O}}_b \cdot \mathbf{R}$ ▷ Re-randomize the basis matrix for O

8: $\quad \mathbf{A} \leftarrow \mathbf{0}_{m\times m}$, $\mathbf{y} \leftarrow \mathbf{0}_m$ ▷ Initialize $\mathbf{A}$ and $\mathbf{y}$

9: $\quad$ **for** $i = 0$ **upto** $m-1$ **do**

10: $\quad\quad \mathbf{A}[i,:] \leftarrow \mathbf{v}^\top \cdot \left(\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)^\top} \;\middle|\; \mathbf{P}_i^{(2)} \right) \cdot \bar{\mathbf{O}}_b$

11: $\quad\quad \mathbf{y}[i] \leftarrow \mathbf{v}^\top \mathbf{P}_i^{(1)} \mathbf{v}$

12: $\quad$ **if** $\mathbf{A}$ is invertible **then**

13: $\quad\quad$ Solve $\mathbf{A}\mathbf{x} = \mathbf{t} - \mathbf{y}$ for $\mathbf{x}$

14: $\quad\quad \mathbf{s} \leftarrow \begin{pmatrix} \mathbf{v} \\ \mathbf{0}_m \end{pmatrix} + \bar{\mathbf{O}}_b \cdot \mathbf{x}$ ▷ $\bar{\mathbf{O}}_b \cdot \mathbf{x}$ cancels the effect of blinding on $\mathbf{s}$

15: $\quad\quad$ Store $\bar{\mathbf{O}}_b$ to persistent storage. ▷ Keep $\bar{\mathbf{O}}_b$ for the next signing operation

16: $\quad\quad$ **return** $\sigma = (\mathbf{s}, \mathsf{salt})$ ▷ σ is identical to the unblinded case

17: **return** $\perp$

randomized in each signing attempt, while other intermediates like $\mathbf{v}$ are ephemeral by construction. The remaining static attack target is $\mathsf{seed_{sk}}$ that is used to sample $\mathbf{v}$ using Shake_{256}. Yet, $\mathsf{seed_{sk}}$ is only used within a single Shake_{256} call, which limits the attack surface. In addition, existing countermeasures like masking the Shake_{256} core can protect $\mathsf{seed_{sk}}$. Alternatively, since the vinegar vector $\mathbf{v}$ can be any randomly selected vector, $\mathsf{seed_{sk}}$ can be replaced by an ephemeral random bitstring of length $\mathsf{sk\_seed\_len}$. The random bitstring is refreshed in each signature computation, yielding genuine signatures under the same public key. With these strategies, we turn the static secret variables into randomized, ephemeral variables, thereby impeding DPA attacks.

In addition, Algorithm 4 shows that most signing operations are unaffected by our blinding, which allows us to easily combine blinding with other existing countermeasures like masking. Also, the blinding itself can be masked by using matrix multiplication gadgets [KNK+25] for the operation in line 7. We note that the blinding of the oil space basis matrix also applies to other round-3 MV signature schemes such as MAYO [BCC+25]. The MAYO scheme is similar to UOV, but optimizes the public key size using an emulsification technique. Given this similarity, our blinding directly applies to MAYO as well.

4.3 Sampling of Random Non-Singular Matrices

The blinding approach introduced in Section 4.2 randomizes the basis matrix by computing $\bar{\mathbf{O}}_b \cdot \mathbf{R}$, with $\mathbf{R}$ sampled from $\mathrm{GL}_m(\mathbb{F}_q)$. Thereby, $\mathbf{R}$ must be non-singular to preserve the rank of $\bar{\mathbf{O}}_b$. However, sampling random non-singular matrices efficiently in resource-constrained hardware is not trivial. In this section, we will investigate two existing approaches for sampling matrices from $\mathrm{GL}_m(\mathbb{F}_q)$ and introduce a hardware-friendly *LU approach* based on the lower-upper decomposition of matrices. To the best of our knowledge, LU-based matrix sampling has not been proposed in the side-channel literature. In addition, sampling random non-singular matrices is not only relevant for our blinded UOV, but also for the masking scheme proposed in [CGZ26]. Therefore, the discussed methods have a

Algorithm 5 Constructing uniformly random non-singular matrix $\mathbf{A} \in \mathrm{GL}_m(\mathbb{F}_q)$ [XDB18]

RandGL(m) ▷ Returns a uniform $\mathbf{R} \xleftarrow{\$} \mathrm{GL}_m(\mathbb{F}_q)$

1: $\mathbf{R} \xleftarrow{\$} \mathrm{GL}_1(\mathbb{F}_q) \cong \mathbb{F}_q^*$ ▷ Random non-zero field element interpreted as 1×1 matrix
2: **for** $i = 2$ **upto** m **do** $\mathbf{R} \leftarrow$ RandExtend$(\mathbf{R})$
3: **return** $\mathbf{R}$

RandExtend$(\mathbf{R})$ ▷ $\mathbf{R} \in \mathrm{GL}_k(\mathbb{F}_q)$; returns uniform $\mathbf{B} \in \mathrm{GL}_{k+1}(\mathbb{F}_q)$

4: $\mathbf{u} \xleftarrow{\$} \mathbb{F}_q^{k+1} \setminus \{\mathbf{0}_{k+1}\}, \quad \mathbf{c} \xleftarrow{\$} \mathbb{F}_q^k$ ▷ Rejection sampling for $\mathbf{u}$
5: $r \leftarrow \min\{\, j \in [k{+}1] : \mathbf{u}[j] \neq 0 \,\}$ ▷ First nonzero index of $\mathbf{u}$
6: $\mathbf{B} \leftarrow \begin{pmatrix} \mathbf{0}_r & 1 & \mathbf{0}_{k-r} \\ \mathbf{R}[:, 0{:}r] & \mathbf{c} & \mathbf{R}[:, r{:}k] \end{pmatrix}$ ▷ Insert unit row and column vector $\mathbf{c}$ at index r
7: **for** $j = r + 1$ **upto** k **do** $\mathbf{B}[:, j] \leftarrow \mathbf{B}[:, j] + \mathbf{u}[j] \cdot \mathbf{B}[:, r]$
8: $\mathbf{B}[:, r] \leftarrow \mathbf{u}[r] \cdot \mathbf{B}[:, r]$
9: **return** $\mathbf{B}$

broader application range in the PQC domain.

4.3.1 Approach 1: Iterative Construction of Random Non-Singular Matrices

The first approach considered is the iterative construction of random non-singular matrices, as proposed in [Ran93] and extended in [XDB18]. This construction returns a uniformly random $\mathbf{R} \xleftarrow{\$} \mathrm{GL}_m(\mathbb{F}_q)$, according to Algorithm 5. The algorithm starts with a non-singular 1×1 matrix. Then, RandExtend() iteratively extends a non-singular $k \times k$ matrix to a $k+1 \times k+1$ matrix until the matrix reaches the targeted dimension m.

However, Algorithm 5 has two problems for efficient UOV hardware. First, line 6 requires data-dependent memory accesses and data movement, since the position of $\mathbf{c}$ in $\mathbf{B}$ depends on the secret index r. Second, the computation of r itself and the updates of $\mathbf{B}$ (lines 5 - 7) must be constant time to prevent timing leakage of r. Thus, supporting Algorithm 5 in constant time would cause runtime and control logic overheads.

4.3.2 Approach 2: Rejection Sampling of Random Non-Singular Matrices

Another approach to sampling $\mathbf{R}$ is rejection sampling. Rejection sampling first samples a random matrix $\mathbf{R} \xleftarrow{\$} \mathbb{F}_q^{m \times m}$ and accepts $\mathbf{R}$ if it is non-singular. Otherwise, $\mathbf{R}$ is rejected, and the procedure is repeated with a freshly sampled $\mathbf{R}$. The rejection probability of a random $\mathbf{R} \xleftarrow{\$} \mathbb{F}_q^{m \times m}$ is $1 - \frac{|\mathrm{GL}_m(\mathbb{F}_q)|}{|\mathbb{F}_q^{m \times m}|} = 1 - \frac{\prod_{i=0}^{m-1}(q^m - q^i)}{q^{m^2}} = 1 - \prod_{j=1}^{m}(1 - q^{-j})$, resulting in a low rejection probability of about 0.4% for UOV-Ip, UOV-III, and UOV-V. For UOV-Is, however, the rejection probability is 6.6% due to the smaller field $\mathbb{F}_{16}$.

The check if $\mathbf{R}$ is non-singular can reuse the Echelon Form (EF) computation in the GE unit, which is already present in our UOV hardware. This makes the rejection sampling approach more efficient than the iterative construction from Section 4.3.1. Yet, the EF computation is a rather slow operation and takes between 19k clock cycles for UOV-Ip and 90k cycles for UOV-V on our hardware. Additional cycles are required for re-sampling a fresh $\mathbf{R}$ after a rejection and for multiplying $\bar{\mathbf{O}}_b \cdot \mathbf{R}$, if $\mathbf{R}$ is non-singular. This matrix multiplication cannot be parallelized with EF and adds between 13.6k and 147k cycles. The resulting signing runtime overhead of rejection sampling-based blinding is between 33k cycles (50% overhead) for UOV-Ip and 237k cycles (46% overhead) for UOV-V. Hence, we propose a more efficient matrix sampling with less runtime overhead in the next subsection.

Table 2: Number of non-singular matrices generated by our LU approach.

Param. Set	q	m	Possible matrices constructed $\lvert\mathrm{LU}_m(\mathbb{F}_q)\rvert$	Share of invertible matrices covered $\lvert\mathrm{LU}_m(\mathbb{F}_q)\rvert/\lvert\mathrm{GL}_m(\mathbb{F}_q)\rvert$
UOV-Ip	256	44	$2^{15,487}$	85%
UOV-Is	16	64	$2^{16,378}$	1.7%
UOV-III	256	72	$2^{41,471}$	76%
UOV-V	256	96	$2^{73,727}$	69%

4.3.3 Approach 3: The LU Approach for Random Non-Singular Matrices

To efficiently sample random non-singular matrices, we propose the so-called *LU approach*. In the LU approach, we sample a random lower-triangular matrix $\mathbf{L} \in \mathbb{F}_q^{m\times m}$ with unit diagonal elements, and a random upper-triangular matrix $\mathbf{U} \in \mathbb{F}_q^{m\times m}$ with non-zero diagonal elements. Then, we compute the randomly sampled matrix $\mathbf{R}$ as

$$\mathbf{R} = \mathbf{L}\cdot\mathbf{U} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{1,0} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{m-1,0} & l_{m-1,1} & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{0,0} & u_{0,1} & \cdots & u_{0,m-1} \\ 0 & u_{1,1} & \cdots & u_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{m-1,m-1} \end{bmatrix}.$$

Note that $l_{i,j} \xleftarrow{\$} \mathbb{F}_q$ for $i > j$, $u_{i,i} \xleftarrow{\$} \mathbb{F}_q^*$, and $u_{i,j} \xleftarrow{\$} \mathbb{F}_q$ for $i < j$. Obviously, $\mathbf{R}$ is non-singular since $\mathbf{L}$ and $\mathbf{U}$ are non-singular and $\mathbf{R}^{-1} = \mathbf{U}^{-1}\cdot\mathbf{L}^{-1}$. However, using our LU approach, $\mathbf{R}$ is not uniformly random over $\mathrm{GL}_m(\mathbb{F}_q)$, since a non-singular matrix $\mathbf{R}$ only permits a LU decomposition if and only if all leading principal minors of $\mathbf{R}$ are non-zero [HJ85]. Due to this non-uniformity, we analyze the statistical properties of $\mathbf{R} = \mathbf{LU}$ in the following. First, we investigate the number of possible $\mathbf{R}$ matrices before examining the probability distribution of individual elements of $\mathbf{R}$.

Number of generated matrices. Let $\mathrm{LU}_m(\mathbb{F}_q) = \{\mathbf{R} \in \mathbb{F}_q^{m\times m} : \mathbf{R} = \mathbf{LU}\}$ be the set of matrices generated by the LU approach over a finite field $\mathbb{F}_q$ as defined above. First, we find $\lvert\mathrm{LU}_m(\mathbb{F}_q)\rvert$, which is the number of possible matrices $\mathbf{R}$ constructed by the LU approach. The matrix $\mathbf{L} \in \mathbb{F}_q^{m\times m}$ is lower-triangular with unit diagonal elements. The $m(m-1)/2$ elements below the diagonal are randomly sampled from $\mathbb{F}_q$. Thus, there are $q^{m(m-1)/2}$ possible $\mathbf{L}$ matrices. Next, we consider $\mathbf{U} \in \mathbb{F}_q^{m\times m}$, which is upper-triangular with non-zero diagonal elements. Thus, there are $(q-1)^m \cdot q^{m(m-1)/2}$ possibilities for $\mathbf{U}$. Since the LU-decomposition of our $\mathbf{R}$ is unique[2], the number of matrices $\mathbf{R} = \mathbf{LU}$ constructed by our LU approach is

$$\lvert\mathrm{LU}_m(\mathbb{F}_q)\rvert = (q-1)^m \cdot q^{\frac{m(m-1)}{2}} \cdot q^{\frac{m(m-1)}{2}} = (q-1)^m \cdot q^{m(m-1)}.$$

Table 2 shows the concrete size of $\lvert\mathrm{LU}_m(\mathbb{F}_q)\rvert$ for the UOV parameter sets. We note that there are at least $2^{15,487}$ possible matrices generated using the LU approach for UOV-Ip. This value reaches up to $2^{73,727}$ matrices for UOV-V, leading to a large search space. Considering the share of matrices generated by the LU approach within all non-singular matrices (right column in Table 2), we see that the LU approach covers at least 69% of $\mathrm{GL}_m(\mathbb{F}_q)$ for UOV-Ip, UOV-III, and UOV-V. Yet, in UOV-Is, the share is significantly lower (1.7%) due to the smaller field. Nevertheless, the vast number of possible matrices makes guesses about $\mathbf{R}$ difficult and thus supports the physical security of our blinding.

[2]Since we fix the diagonal values of $\mathbf{L}$ to be 1, the LU decomposition of $\mathbf{R}$ is unique, if it exists [HJ85].

Internal probabilities of R. After examining the size $|\mathrm{LU}_m(\mathbb{F}_q)|$, we now focus on the internal probability distribution of the elements $r_{i,j} = \mathbf{R}[i,j]$. For this investigation, we assume that all values of $l_{i,j} = \mathbf{L}[i,j]$, $i > j$ and $u_{i,j} = \mathbf{U}[i,j]$, $i \leq j$ are unknown to the attacker. Note that this assumption aligns with our DPA threat model. We show that

$$\forall i,j \in [m], i \neq j : P(r_{i,j} = x) = q^{-1} \text{ for any } x \in \mathbb{F}_q \tag{2}$$

and

$$\forall i \in [m] : P(r_{i,i} = x) = \begin{cases} \frac{q^i-1}{q^{i+1}} & \text{if } x = 0 \\ \frac{q^{i+1}-q^i+1}{q^{i+2}-q^{i+1}} & \text{if } x \in \mathbb{F}_q^* \end{cases} \tag{3}$$

in the following. We start with Lemma 1 proving Equation (2), and then show Equation (3) in Lemmas 2 and 3.

Lemma 1. *Let $\mathbf{R} = \mathbf{LU} \in \mathbb{F}_q^{m \times m}$ be a matrix generated with the LU approach. The probability that $r_{i,j} = \mathbf{R}[i,j]$ with $i,j \in [m], i \neq j$, takes the value $x \in \mathbb{F}_q$ is $P(r_{i,j} = x) = q^{-1}$. This means that all non-diagonal elements $\mathbf{R}$ appear uniformly random over $\mathbb{F}_q$.*

Proof. Since $\mathbf{L}$ and $\mathbf{U}$ are lower and upper-triangular, respectively, the computation of $r_{i,j}$ for $i < j$ can be written as

$$r_{i,j} = \sum_{k=0}^{i} l_{i,k} \cdot u_{k,j} = l_{i,i} \cdot u_{i,j} + \sum_{k=0}^{i-1} l_{i,k} \cdot u_{k,j}.$$

Note that $l_{i,i} = 1$ by construction. Since $u_{i,j}$ is uniformly random over $\mathbb{F}_q$ and independent of the remaining sum, the involved addition of $u_{i,j}$ randomizes the result $r_{i,j}$. Therefore, $r_{i,j}$ appears as uniformly random over $\mathbb{F}_q$. Similarly, the computation of $r_{i,j}$ for $i > j$ is

$$r_{i,j} = \sum_{k=0}^{j} l_{i,k} \cdot u_{k,j} = l_{i,j} \cdot u_{j,j} + \sum_{k=0}^{j-1} l_{i,k} \cdot u_{k,j}.$$

Since $u_{j,j}$ is uniformly chosen from $\mathbb{F}_q^*$, it cannot be zero. Therefore, multiplying $l_{i,j}$ (which is uniformly random over $\mathbb{F}_q$) by a non-zero $u_{j,j}$ yields a uniformly random result. This randomizes the remaining sum, and $r_{i,j}$ appears uniformly random over $\mathbb{F}_q$. □

Lemma 2. *Let a_k and b_k be uniformly random elements from $\mathbb{F}_q$ for $k \in [i]$, $i > 0$. Then, the probability that the sum $\sum_{k=0}^{i-1} a_k b_k$ is zero is $P(\sum_{k=0}^{i-1} a_k b_k = 0) = \frac{q^i+q-1}{q^{i+1}}$.*

Proof. First, we note that there are more combinations of $a_k b_k$ evaluating to 0 than for other values $x \in \mathbb{F}_q^*$. Thus, $P(a_k b_k = 0) > P(a_k b_k = x \in \mathbb{F}_q^*)$. Yet, the number of combinations of $a_k b_k$ resulting in a non-zero value x is the same for all $x \in \mathbb{F}_q^*$, leading to $P(a_k b_k = x \in \mathbb{F}_q^*) = P(a_k b_k = y \in \mathbb{F}_q^*)$ for any pair of $x, y \in \mathbb{F}_q^*$. This further leads to $P(a_k b_k = x \in \mathbb{F}_q^*) = \frac{1}{|\mathbb{F}_q^*|}(1 - P(a_k b_k = 0))$.

Using the above observations, we now show Lemma 2 by induction. First, we consider the base case where $i = 1$, leading to

$$P\left(\sum_{k=0}^{0} a_k b_k = 0\right) = \frac{q^1 + q - 1}{q^{1+1}} = \frac{1}{q} + \frac{1}{q} - \frac{1}{q^2}.$$

This can easily be verified since the right-hand side of the above equation describes all combinations of $a_0, b_0 \in \mathbb{F}_q$, where $a_0 \cdot b_0 = 0$. This is the case if and only if $a_0 = 0$ or $b_0 = 0$, since $a_0, b_0 \in \mathbb{F}_q$ is a field.

Next, we consider $i > 1$ and obtain

$$P\left(\sum_{k=0}^{i-1} a_k b_k = 0\right) = P\left(\sum_{k=0}^{i-2} a_k b_k = -a_{i-1}b_{i-1}\right) =$$
$$= P(a_{i-1}b_{i-1} = 0) \cdot P\left(\sum_{k=0}^{i-2} a_k b_k = 0\right) + P(a_{i-1}b_{i-1} \neq 0) \cdot P\left(\sum_{k=0}^{i-2} a_k b_k = x \in \mathbb{F}_q^*\right).$$

With $P(\sum_{k=0}^{i-2} a_k b_k = x \in \mathbb{F}_q^*) = \frac{1}{|\mathbb{F}_q^*|}(1 - P(\sum_{k=0}^{i-2} a_k b_k = 0)) = \frac{1}{q-1}(1 - \frac{q^{i-1}+q-1}{q^i})$ from above, we get the result

$$P\left(\sum_{k=0}^{i-1} a_k b_k = 0\right) = \frac{2q-1}{q^2}\frac{q^{i-1}+q-1}{q^i} + \frac{q^2-2q+1}{q^2}\frac{1}{q-1}\frac{q^i - q^{i-1} - q + 1}{q^i} = \frac{q^i+q-1}{q^{i+1}}.$$

□

Lemma 3. *Let $\mathbf{R} = \mathbf{LU} \in \mathbb{F}_q^{m \times m}$ be a matrix generated with the LU approach. The probability that $r_{i,i}$ with $i \in [m]$, is 0 is $P(r_{i,i} = 0) = \frac{q^i-1}{q^{i+1}}$. Furthermore, the probability that $r_{i,i}$ equals some $x \in \mathbb{F}_q^*$ is $P(r_{i,i} = x) = \frac{q^{i+1}-q^i+1}{q^{i+2}-q^{i+1}}$.*

Proof. The computation of the diagonal elements $r_{i,i}$ can be done via

$$r_{i,i} = \sum_{k=0}^{i} l_{i,k} \cdot u_{k,i} = \underbrace{l_{i,i}}_{=1} \cdot u_{i,i} + \sum_{k=0}^{i-1} l_{i,k} \cdot u_{k,i},$$

with $u_{i,i} \in \mathbb{F}_q^*$ a uniformly random non-zero element that is independent of the remaining sum $\sum_{k=0}^{i-1} l_{i,k} \cdot u_{k,i}$. We first consider the case $P(r_{i,i} = 0)$ and use

$$P(r_{i,i} = 0) = P\left(\sum_{k=0}^{i-1} l_{i,k} \cdot u_{k,i} = -u_{i,i}\right).$$

Since each value of $u_{i,i} \in \mathbb{F}_q^*$ has the same probability $P(u_{i,i} = x \in \mathbb{F}_q^*) = 1/|\mathbb{F}_q^*|$, we have

$$P(r_{i,i} = 0) = P\left(\sum_{k=0}^{i-1} l_{i,k} \cdot u_{k,i} = -u_{i,i}\right) = \frac{1}{|\mathbb{F}_q^*|} \cdot \left(1 - P\left(\sum_{k=0}^{i-1} l_{i,k} \cdot u_{k,i} = 0\right)\right). \quad (4)$$

Using Lemma 2 in Equation (4) gives

$$P(r_{i,i} = 0) = \frac{1}{q-1} \cdot \left(1 - \frac{q^i+q-1}{q^{i+1}}\right) = \frac{1}{q-1} \cdot \frac{q^{i+1} - q^i - q + 1}{q^{i+1}} = \frac{q^i - 1}{q^{i+1}},$$

which proves Equation (3) for $x = 0$. Next, we consider the case $P(r_{i,i} = x)$ for non-zero $x \in \mathbb{F}_q^*$. Since all non-zero results for $r_{i,i}$ are equally likely, we obtain

$$P(r_{i,i} = x \in \mathbb{F}_q^*) = (1 - P(r_{i,i} = 0)) \cdot P(r_{i,i} = x \in \mathbb{F}_q^* \,|\, r_{i,i} \neq 0) = \frac{q^{i+1} - q^i + 1}{q^{i+2} - q^{i+1}}.$$ □

The results in Equations (2) and (3) show that all off-diagonal elements $r_{i,j} = \mathbf{R}[i,j]$ with $i \neq j$ appear uniformly random over $\mathbb{F}_q$. However, the diagonal elements $r_{i,i}$ have a bias towards being non-zero. This bias is strongest for $i = 0$, where $r_{0,0}$ is definitely non-zero, and becomes less pronounced for increasing i. For $i > 4$, the bias in the diagonal

Table 3: Operation scheduling and datapath operand assignment for blinded UOV signing.

	UOV signing operations executed from left to right (Algorithm 4)							
	Shake	$\bar{\mathbf{O}}_b \cdot \mathbf{L}$	$\bar{\mathbf{O}}_b\mathbf{L} \cdot \mathbf{U}$	**vPv**	**vP**	**vP**$\bar{\mathbf{O}}_b$	**GE**	$\mathbf{v} + \bar{\mathbf{O}}_b\mathbf{x}$
Step	0	1	2	3	4	5	6	7
Alg. line	2, 5	7	7	11[a]	10[a]	10[a]	13	14
in_a	-	AES	AES	BRAM_vs	BRAM_vs	BRAM_O	BRAM_A/_ty/_T	BRAM_ty
in_b	-	1	1	BRAM_vs	1	1	piv_inv	1
in_c	-	BRAM_O	BRAM_T	AES	AES	BRAM_T	BRAM_A/_ty/_T	BRAM_O
preload	-	0	0	BRAM_ty	0	0	BRAM_A/_ty/_T	BRAM_vs
out	BRAM_ty/_vs	BRAM_T	BRAM_O	BRAM_ty/_T	BRAM_T	BRAM_A/_T	BRAM_A/_ty/_T	BRAM_vs

[a]Executed once per slice. UOV-Ip, UOV-III, and UOV-V have $\lceil m/16 \rceil = 3$, 5, and 6 slices, respectively.

elements is already very small (at most 2^{-23} for UOV-Is and 2^{-48} for other UOV parameter sets). The probability that a diagonal element $r_{i,i}$ takes any value of $\mathbb{F}_q^*$ is uniform over $\mathbb{F}_q^*$.

In conclusion, our LU approach samples matrices $\mathbf{R} \xleftarrow{\$} \mathrm{LU}_m(\mathbb{F}_q) \subset \mathrm{GL}_m(\mathbb{F}_q)$ uniformly at random from $\mathrm{LU}_m(\mathbb{F}_q)$, and offers a low-overhead side-channel countermeasure via blinding UOV's secret oil space. Compared to rejection sampling (Section 4.3.2), LU-based blinding has less runtime overhead of 21% to 29% for different security levels. We again emphasize that the proposed blinding with the LU approach is a trade-off between physical security and efficiency, which is attractive for resource-constrained devices.

4.4 Hardware Support for LU-based Blinding and Operation Scheduling

The key strength of our LU-based blinding is its computational efficiency. The blinding only reuses existing datapath elements and BRAM memories. Thus, the area overhead of blinding is marginal. In addition, we implement the LU approach for sampling $\mathbf{R} = \mathbf{LU}$ by reutilizing the $\mathsf{AES}_{128}\mathsf{CTR}$ modules with a dedicated secret seed for blinding. This allows us to generate the components of $\mathbf{L}$ and $\mathbf{U}$ on-the-fly and directly multiply them by $\bar{\mathbf{O}}_b$.

To illustrate the computation flow of our blinded signing, we detail the operation scheduling and the datapath operand assignment in Table 3. The signing starts with the I/O module (cf. Figure 1) loading the message and the salt into the upper half of BRAM_vs. Then, Shake_{256} samples the target vector $\mathbf{t}$ into BRAM_ty and the vinegar vector $\mathbf{v}$ into the lower half of BRAM_vs (step 0 in Table 3). Next, steps 1 and 2 re-randomize $\bar{\mathbf{O}}_b$ following the LU approach: we sample $\mathbf{L}$ and $\mathbf{U}$ on-the-fly from $\mathsf{AES}_{128}\mathsf{CTR}$ and multiply them with $\bar{\mathbf{O}}_b$. Then, we apply slicing and perform the $\mathbf{vPv}$, $\mathbf{vP}$, and $\mathbf{vP}\bar{\mathbf{O}}_b$ steps $\lceil m/16 \rceil$ times in a loop[3], where each loop iteration computes over the 16 different $\mathbf{P}_i$ matrices of a slice. The $\mathbf{vPv}$ operation (step 3) computes the quadratic polynomial by reading two elements from BRAM_vs using two read ports. Next, we perform $\mathbf{vP}$ to reduce data blowup and only then compute $\mathbf{vP}\bar{\mathbf{O}}_b$. Finally, the GE (step 6) tries to solve the linear system $\mathbf{Ax} = \mathbf{t} - \mathbf{y}$. As explained in Section 3.4, the data for $\mathbf{A}$ and $\mathbf{t} - \mathbf{y}$ is mirrored in BRAM_T to satisfy the memory port requirements of GE. If the system is solvable, the signature is computed in step 7 and stored in BRAM_vs. Finally, the host reads the signature via the I/O module. Using this operation flow, we fully pipeline the matrix operations and the blinding to reduce the signing runtime.

5 Results and Comparison

We implement our UOV accelerator with the LU-based blinding countermeasure on a low-end Artix7 FPGA (xc7a100tftg256-2). We use VitisHLS and SystemVerilog for hardware description and synthesize, place, and route our design via Xilinx Vivado/Vitis 2022.2. We

[3]We abuse notation for brevity here. $\mathbf{vPv}$ stands for the operation in Algorithm 4, line 11, $\mathbf{vP}$ and $\mathbf{vP}\bar{\mathbf{O}}_b$ stands for Algorithm 4, line 10, and $\mathbf{v} + \bar{\mathbf{O}}_b\mathbf{x}$ stands for Algorithm 4, line 14

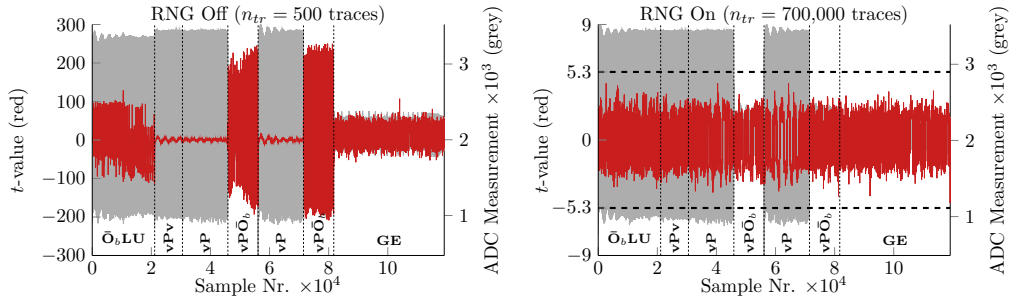


Figure 4: Power traces (grey) and non-specific fixed versus random TVLA (red) of our LU-blinded signing. Our design with RNG On does not leak for n_{tr} = 700,000 traces, although the unprotected case (RNG Off) already leaks at n_{tr} = 500 traces.

verified the functional correctness and side-channel resistance by running our design on the real FPGA. This section presents the obtained experimental and implementation results.

5.1 TVLA Assessment

To show the effectiveness of our blinding countermeasure, we measure the power consumption of the FPGA and perform a test vector leakage assessment (TVLA). Our setup consists of a NewAE ChipWhisperer CW305 board featuring our target Artix7 FPGA [Inc26b]. For the power measurements, we clock the FPGA with an external 10 MHz clock and collect the power traces with a ChipWhisperer Husky [Inc26a]. Husky's ADC has 12-bit resolution, and we take 4 samples per FPGA clock cycle. Similar to prior work [KNK+25], we use a toy parameter set of $q = 256$, $m = 32$, and $n = 80$ to accelerate trace collection.

Our TVLA assessment follows the non-specific fixed versus random approach [GJJR11], where we randomly interleave signing operations with a LU-blinded fixed $\bar{\mathbf{O}}_b$ matrix and LU-blinded random $\bar{\mathbf{O}}$ matrices. All other inputs, like the message, salt, or vinegar vectors, remain fixed to only cover leakage related to $\bar{\mathbf{O}}_b$. We then compute Welch's t-test for each point in the power trace to detect leakage. Following [DZD+18] and with our trace length of ≈120k samples, a t-value within a ±5.3 boundary indicates that the fixed and random trace sets cannot be distinguished with high confidence (significance level $\alpha = 0.01$).

The measurement results are shown in Figure 4 for disabled blinding (RNG Off, left) and active blinding (RNG On, right). To ensure consistent timing, we fix $\mathbf{R}$ to be the identity matrix $\mathbf{I}_m$ in the RNG Off case. Considering the t-value for RNG Off (red in Figure 4 left), the unprotected design already shows pronounced leakage at $n_{tr} = 500$ traces per set, with the t-value far larger than the ±5.3 threshold. This leakage of the unprotected design is strongest during operations involving $\bar{\mathbf{O}}_b$, as expected. However, with active blinding (RNG On, Figure 4 right), even the t-value for n_{tr} = 700,000 traces is well within the ±5.3 range and thus does not indicate leakage. Note that these results use our LU-based blinding approach from Section 4.3.3, which has a small bias in the generated non-singular matrices. This bias does not cause leakage in our t-test, and we obtain similar t-test results for blinding with uniformly random $\mathbf{R}$. Hence, LU-based blinding increases the complexity of DPA attacks while avoiding large runtime and area penalties.

5.2 Implementation Results

We present our implementation results and start with the timing of individual suboperations in our UOV accelerator. As shown in Table 4, UOV's signing involves various suboperations including Shake_{256}, matrix operations, and solving the linear equation system using GE. Shake_{256} only consumes a minor fraction of latency, while the matrix operations (**vPv**, **vP**,

Table 4: Latency breakdown of UOV signing (Algorithm 4) in clock cycles. The message length is 16 bytes, and $\mathbf{A}$ is non-singular in the first signing attempt.

	Shake	$\bar{\mathbf{O}}_b$**LU**	**vPv**	**vP**	**vP**$\bar{\mathbf{O}}_b$	**GE**	**v+**$\bar{\mathbf{O}}_b$**x**	**Total Latency**		
Alg. line	2, 5	7	11[a]	10[a]	10[a]	13	14	Plain	Blinded	Ovhd.[b]
UOV-Ip	136	13.9k	3× 2.3k	3× 7.6k	3× 4.9k	19.4k	335	64,586	78,446	1.21×
UOV-III	136	63.1k	5× 6.3k	5× 20.6k	5× 13.2k	52.0k	891	253,908	316,980	1.25×
UOV-V	170	149.0k	6× 11.0k	6× 36.1k	6× 23.4k	90.3k	1,563	515,468	664,460	1.29×

[a]Executed once per slice. UOV-Ip, UOV-III, and UOV-V have $\lceil m/16 \rceil = 3$, 5, and 6 slices. [b]Blinding overhead

and $\mathbf{vP}\bar{\mathbf{O}}_b$) dominate the latency. These matrix operations construct the linear equation system using our slicing approach, which iteratively computes $\mathbf{A}$ to save memory. For UOV-Ip, UOV-III, and UOV-V, the combined latency of $\mathbf{vPv}$, $\mathbf{vP}$, and $\mathbf{vP}\bar{\mathbf{O}}_b$ is 45k, 201k, and 423k clock cycles, causing 68%, 79%, and 82% of the overall unblinded signing latency. Another major step in signing is the GE, which causes 30%, 20%, and 18% of the unblinded signing latency for UOV-Ip, UOV-III, and UOV-V, respectively. Our blinding from Section 4 randomizes the basis matrix $\bar{\mathbf{O}}_b$ by multiplying a random $\mathbf{R} \in \mathrm{LU}_m(\mathbb{F}_q)$. Blinding samples $\mathbf{R} = \mathbf{LU}$ on-the-fly using $\mathsf{AES}_{128}\mathsf{CTR}$ and takes additional 14k, 63k, and 149k clock cycles for UOV-Ip, UOV-III, and UOV-V, respectively. This corresponds to an overall signing latency overhead between 1.21× and 1.29×, which is less than the $\approx 1.5\times$ overhead when using rejection sampling for $\mathbf{R}$ (Section 4.3.2).

The verification consists of two main operations, which are hashing (line 1 in Algorithm 2) and the computation $\mathbf{s}^\top \mathbf{P}_i \mathbf{s}$ (line 3 in Algorithm 2). Similar to signing, hashing in verification is very fast and consumes less than 100 clock cycles for all parameter sets. For computing $\mathbf{s}^\top \mathbf{P}_i \mathbf{s}$, we generate $\mathbf{P}_i^{(1)}$ and $\mathbf{P}_i^{(2)}$ on-the-fly, and stream $\mathbf{P}_i^{(3)}$ into the co-processor as explained in Section 3.5. This allows a fully pipelined computation and a verification latency of 19k, 85k, and 179k cycles for UOV-Ip, UOV-III, and UOV-V.

Area Consumption. Table 5 shows the area breakdown of our design. The whole UOV accelerator, which supports UOV-Ip, UOV-III, and UOV-V parameter sets, consumes 34k lookup tables (LUTs), 26k registers (REGs), 14 digital signal processor slices (DSPs), and 22 block RAMs (BRAMs). Each BRAM has a capacity of 36 kbit. Within our design, the Shake_{256} core and the Data Bus cause the major LUT consumption with about 11k LUTs each. The LUT consumption of the Data Bus is due to the multiplexing of data signals across the different submodules, as shown in Figure 1. The two fully unrolled and pipelined 10-round $\mathsf{AES}_{128}\mathsf{CTR}$ modules are responsible for on-the-fly data sampling and realignment. The AES subsystem consumes 6.6k LUTs and 5 DSPs, where the DSPs are for $\mathsf{AES}_{128}\mathsf{CTR}$ index computation. Finally, the fully pipelined datapath module consumes about 3k LUTs and 1.6k REGs. The area consumption in Table 5 is largely independent of the blinding support since our blinding re-uses existing datapath and $\mathsf{AES}_{128}\mathsf{CTR}$ components. Moreover, we reach a low BRAM consumption of just 22 BRAMs, even for the large UOV-V parameter set. This is due to our slicing technique and on-the-fly data sampling. When UOV-V support is not required, the BRAM count can be reduced to 18 and 14 BRAMs for UOV-Ip+UOV-III and UOV-Ip-only support, respectively.

5.3 Comparison to Related Work

We now compare our design to related hardware accelerators. This comparison considers the existing hardware accelerators for UOV and MAYO since these schemes are closely related. Table 6 presents the key benchmarks, including area consumption, signing and verification latency. The comparison also includes the area-time-product for signing (ATP) [YCH22]. We note that all works in Table 6 use comparable Xilinx 7-series FPGAs, which allows a fair comparison. Since the related works do not include blinding or masking

Table 5: Area breakdown of our submodules and their relative share of the total area.

Module	LUTs		REGs		DSPs		BRAMs	
Artix7-100t FPGA	63,400		126,800		240		135	
Memory Subsystem	780	2.3%	389	1.5%	-	-	22	100%
Shake_{256}	11,707	34.0%	10,426	39.7%	-	-	-	-
2× $\mathsf{AES}_{128}\mathsf{CTR}$ (10 rounds)	6,657	19.3%	5,246	20.0%	5	35.7%	-	-
Datapath	2,961	8.6%	1,632	6.2%	-	-	-	-
GE Ctrl. + Field Inversion	1,071	3.1%	1,725	6.6%	4	28.6%	-	-
Overall Ctrl. + Data Bus	11,034	32.0%	3,611	13.8%	5	35.7%	-	-
I/O	228	0.7%	3,214	12.2%	-	-	-	-
Total	**34,438**	**100%**	**26,243**	**100%**	**14**	**100%**	**22**	**100%**

Table 6: Comparison to related hardware acceleration works using pipelined AES.

Work	Scheme	FPGA	Freq. MHz	Sign kcc	Verify kcc	LUTs k	REGs k	DSPs	BRAMs	ATP[a]
[SMA+24]	MAYO$_1$	Z-7020	100	2,867	–	21	13	11	129	1,142
[HSK+24]	MAYO$_1$	xc7k410t	100	31	5	112	38	2	45.5	4
	MAYO$_3$	xc7k410t	100	66	9	169	60	2	96	19
	MAYO$_5$	xc7k410t	100	112	15	246	83	2	194.5	66
[BCH+23]	UOV-Ip	Z-7000	91.8	303	61	38	26	2	48	48
	UOV-III	xc7a200t	94.1	1,200	196	43	32	4	184.5	712
	UOV-V	xc7a200t	92.6	2,646	364	83	41	4	359	3,092
Our Unblinded/ Blinded	UOV-Ip	xc7a100t	100	65 / 78	19					5 / 6
	UOV-III	xc7a100t	100	254 / 317	85	34	26	14	22	20 / 26
	UOV-V	xc7a100t	100	515 / 664	179					42 / 54

[a] Area-time product: $(\#\text{LUTs} + \#\text{REGs} + 100 \cdot \#\text{DSPs} + 300 \cdot \#\text{BRAMs}) \cdot$ signing latency in seconds [YCH22]

countermeasures, we compare our latency for both blinded and unblinded signing.

We start the comparison with HaMAYO [SMA+24], which proposes a hardware implementation of MAYO's signing and key generation for NIST security level 1. The authors of HaMAYO target the round-1 MAYO parameters with $\mathbb{F}_{31}$ and include lightweight shuffling and fault attack countermeasures. HaMAYO reports a signing latency of 2.87M clock cycles, whereas our design only needs 65k and 78k clock cycles for unblinded and blinded UOV-Ip, respectively. This shows our signing speedup of 37× to 44×, while we reduce the BRAM consumption by 5.9×. Next, we consider [HSK+24], which implements the round-2 MAYO scheme including signing, verification, and key generation functionality. The authors of [HSK+24] focus on high performance, which is reflected in their low signing and verification latency, outperforming our design. Yet, the area and memory consumption in [HSK+24] are clearly higher than in our work. Despite the different optimization goals, the signing ATP of [HSK+24] and our ATP (without blinding) are similar for security levels 1 and 3. For security level 5, our unprotected ATP is 1.6× lower. Finally, we compare to [BCH+23] targeting UOV hardware acceleration. The authors of [BCH+23] do not use on-the-fly data generation and store all $\{\mathbf{P}_i\}_{i\in[m]}$ matrices on chip. This explains their high BRAM count, which is between 2.2× and 16× higher than in our runtime-flexible UOV architecture. In addition, we reduce the signing latency by 4.7× (UOV-Ip) to 5.1× (UOV-V) in the unblinded case. In the blinded case, our signing speedup is between 3.8× and 4×, and our verification speedup is between 3.2× for UOV-Ip and 2× for UOV-V. Considering the ATP of unprotected signing, we outperform [BCH+23] by at least 9.2×.

The comparison to related hardware accelerators shows the benefits of our low-memory yet performant design. We significantly reduce the BRAM consumption, which is crucial for efficient hardware deployment of UOV. At the same time, we maintain competitive signing and verification performance, even when including the blinding countermeasure.

6 Conclusion

In this paper, we investigated lightweight and memory-efficient hardware acceleration of the UOV signature scheme. We presented architecture-level optimizations such as slicing, on-the-fly data sampling, and streamlined data realignment, which significantly reduce the critical memory consumption of UOV. Moreover, we proposed a blinding countermeasure based on equivalent keys that involves random non-singular matrices. Since sampling non-singular matrices efficiently in hardware is difficult, we discussed different existing approaches and introduced our hardware-friendly LU approach. The resulting blinding countermeasure is less powerful than masking, but only causes a 30% runtime overhead for signing while reusing existing datapath components. Thereby, blinding allows trade-offs between physical security and performance and can easily be combined with other countermeasures such as masking or shuffling. Our implementation results show up to 16× lower memory consumption and competitive performance compared to existing UOV and MAYO accelerators. Finally, we note that the insights of this paper not only apply to UOV but also extend to other MV schemes such as MAYO. Therefore, our contributions support efficient and lightweight hardware acceleration of MV signature schemes.

Acknowledgement

We used generative AI (Claude Opus 4.8) for basic editing, grammar, and spell checking, as well as for improving the formatting and layout of tables and figures.

This project has received funding from the European Union's Horizon Europe research and innovation programme via the Rigoletto project (grant agreement ID: 101194371). It was also partially supported by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.

We thank Tobias Schneider and Melissa Azouaoui from NXP Semiconductors for the helpful technical discussions and the valuable feedback. We also thank Florian Hirner from Graz University of Technology for hardware-related discussions and for carefully reviewing the paper.

References

[AAC+22] 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. Nist ir 8413-upd1 – status report on the third round of the nist post-quantum cryptography standardization process. NIST Post-Quantum Cryptography Standardization, 2022. https://doi.org/10.6028/NIST.IR.8413-upd1.

[ABC+26] Gorjan Alagic, Maxime Bros, Pierre Ciadoux, Quynh Dang, Thinh Hung Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Hamilton Silberg, Daniel Smith-Tone, and Noah Waller. Nist internal report 8610 – status report on the second round of the additional digital signature schemes for the nist post-quantum cryptography standardization process. NIST Post-Quantum Cryptography Standardization: Additional Digital Signature Schemes, 2026. https://doi.org/10.6028/NIST.IR.8610.

[ACK+23] Thomas Aulbach, Fabio Campos, Juliane Krämer, Simona Samardjiska, and Marc Stöttinger. Separating oil and vinegar with a single trace: Side-channel assisted kipnis-shamir attack on uov. *IACR Transactions on Cryptographic*

Hardware and Embedded Systems, 2023(3):221–245, Jun. 2023. https://tches.iacr.org/index.php/TCHES/article/view/10962.

[ACK25] Thomas Aulbach, Fabio Campos, and Juliane Krämer. Sok: On the physical security of uov-based signature schemes. In Ruben Niederhagen and Markku-Juhani O. Saarinen, editors, *Post-Quantum Cryptography*, pages 199–231, Cham, 2025. Springer Nature Switzerland.

[AMD21] AMD Inc. 7 series product tables and product selection guide (xmp101). AMD Technical Information Portal, 2021. https://docs.amd.com/v/u/en-US/7-series-product-selection-guide, Accessed in July 2026.

[BCC+24] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. Nibbling mayo: Optimized implementations for avx2 and cortex-m4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):252–275, Mar. 2024.

[BCC+25] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. Mayo specification round 2 version. NIST Post-Quantum Cryptography Standardization: Additional Digital Signature Schemes, Round 2 Submission, 2025. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/mayo-spec-round2-web.pdf.

[BCD+25] Ward Beullens, Ming-Shing Chen, Jintai Ding, Boru Gong, Matthias J. Kannwischer, Jacques Patarin, Bo-Yuan Peng, Dieter Schmidt, Cheng-Jhih Shih, Chengdong Tao, and Bo-Yin Yang. UOV: Unbalanced oil and vinegar – algorithm specifications and supporting documentation, version 2.0. NIST Post-Quantum Cryptography Standardization: Additional Digital Signature Schemes, Round 2 Submission, 2025. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/uov-spec-round2-web.pdf, Accessed in July 2026.

[BCH+23] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. Oil and vinegar: Modern parameters and implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):321–365, Jun. 2023.

[BDSK25] Sven Bauer, Fabrizio De Santis, and Kristjane Koleci. Fault attacks against uov-based signatures. In *2025 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pages 52–60, 2025.

[Beu22] Ward Beullens. Breaking rainbow takes a weekend on a laptop. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 464–479, Cham, 2022. Springer Nature Switzerland.

[CGZ26] Jean-Sébastien Coron, François Gérard, and Bowen Zhang. Masked solving of linear equations system and application to uov signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(2):51–72, Apr. 2026.

[CNU26] Chung Chen, Ruben Niederhagen, and Yeong-Luh Ueng. An fpga accelerator for extended gaussian elimination in wave. *IEEE Embedded Systems Letters*, pages 1–1, 2026.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 164–175, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[DZD+18] A. Adam Ding, Liwei Zhang, Francois Durvaux, Francois-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications*, pages 105–122, Cham, 2018. Springer International Publishing.

[FG18] Ahmed Ferozpuri and Kris Gaj. High-speed fpga implementation of the nist round 1 rainbow signature scheme. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2018.

[FI26] Hiroki Furue and Yasuhiko Ikematsu. Key recovery attacks on UOV using p^ℓ-truncated polynomial rings. Cryptology ePrint Archive, Paper 2026/298, 2026. https://eprint.iacr.org/2026/298.

[GJCJ23] Naina Gupta, Arpan Jati, Anupam Chattopadhyay, and Gautam Jha. Lightweight hardware accelerator for post-quantum digital signature crystals-dilithium. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(8):3234–3243, 2023.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side channel resistance validation, 2011. https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf, Accessed in July 2026.

[HJ85] Roger A. Horn and Charles R. Johnson. *Canonical forms*, pages 119–166. Cambridge University Press, 1985. https://www.cambridge.org/core/services/aop-cambridge-core/content/view/18208C572F905261855B1F24C22C8343/9780511810817c4_p119-166_CBO.pdf/canonical_forms.pdf.

[HSK+24] Florian Hirner, Michael Streibl, Florian Krieger, Ahmet Can Mert, and Sujoy Sinha Roy. Whipping the multivariate-based mayo signature scheme using hardware platforms. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, pages 3421–3435, New York, NY, USA, 2024. Association for Computing Machinery.

[Inc26a] NewAE Technology Inc. Chipwhisperer husky, 2026. https://chipwhisperer.readthedocs.io/en/latest/Capture/ChipWhisperer-Husky.html, Accessed in July 2026.

[Inc26b] NewAE Technology Inc. Cw305 artix fpga target product datasheet, 2026. https://media.newae.com/datasheets/NAE-CW305_datasheet.pdf, Accessed in July 2026.

[JD26] Sönke Jendral and Elena Dubrova. Single-trace side-channel attacks on mayo exploiting leaky modular multiplication. In *Proceedings of the 2025 1st Workshop on Quantum-Resistant Cryptography and Security*, QRSEC '25, pages 21–30, New York, NY, USA, 2026. Association for Computing Machinery.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[KNK+25] Suparna Kundu, Quinten Norga, Angshuman Karmakar, Uttam Kumar Ojha, Anindya Ganguly, and Ingrid Verbauwhede. muov: Masking the unbalanced oil and vinegar digital signature scheme at first- and higher-order. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, pages 1994–2008, New York, NY, USA, 2025. Association for Computing Machinery.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[NC24] National Institute of Standards and Technology (NIST) and David Cooper. Stateless hash-based digital signature standard, 2024. https://doi.org/10.6028/NIST.FIPS.205.

[NDL+24] National Institute of Standards and Technology (NIST), Thinh Dang, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, and Angela Robinson. Module-lattice-based digital signature standard, 2024. https://doi.org/10.6028/NIST.FIPS.204.

[NKZ+26] Ziying Ni, Ayesha Khalid, Zhaoyu Zhang, Yijun Cui, Weiqiang Liu, and Máire O'Neill. Lighthd: A lightweight and high-performance hardware accelerator of crystals-dilithium. *IEEE Transactions on Computers*, 75(5):2007–2019, 2026.

[PSN22] David Pokorný, Petr Socha, and Martin Novotný. Equivalent keys: Side-channel countermeasure for post-quantum multivariate quadratic signatures. *Electronics*, 11(21), 2022. https://doi.org/10.3390/electronics11213607.

[Ran93] Dana Randall. Efficient generation of random nonsingular matrices. *Random Structures & Algorithms*, 4(1):111–118, 1993.

[Ran26] Lars Ran. Wedges, oil, and vinegar: An analysis of uov in the exterior algebra. In *Advances in Cryptology - EUROCRYPT 2026: 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10-14, 2026, Proceedings, Part IV*, pages 363–388, Berlin, Heidelberg, 2026. Springer-Verlag.

[SMA+24] Oussama Sayari, Soundes Marzougui, Thomas Aulbach, Juliane Krämer, and Jean-Pierre Seifert. Hamayo: A fault-tolerant reconfigurable hardware implementation of the mayo signature scheme. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 240–259. Springer, 2024.

[WP05] Christopher Wolf and Bart Preneel. Large superfluous keys in multivariate quadratic asymmetric systems. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005*, pages 275–287, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[WP26] Thom Wiggers and PQShield. Nist signatures zoo. online, June 2026. https://pqshield.github.io/nist-sigs-zoo/, Accessed in July 2026.

[XDB18] Sebastià Xambó-Descamps and Narcís Sayols Baixeras. Constructing random invertible matrices over a finite field. Mathematical Notes, 2018. https://web.mat.upc.edu/sebastia.xambo/PyECC/MNs/PyECC-random-invertible-matrices.pdf, Accessed in July 2026.

[YCH22] Zewen Ye, Ray C. C. Cheung, and Kejie Huang. Pipentt: A pipelined number theoretic transform architecture. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(10):4068–4072, 2022.