A High-Speed Hardware Accelerator for QR-UOV Signature Scheme

Renma Sugai[1], Hiroshi Amagasa[1], Rei Ueno[2,1] and Naofumi Homma[1]

[1] Research Institute of Electrical Communication/Graduate School of Engineering, Tohoku University, 2–1–1 Katahira, Aoba-ku, Sendai-shi, 980-8577, Japan, sugai.renma.p1@dc.tohoku.ac.jp, hiroshi.amagasa.q7@dc.tohoku.ac.jp, naofumi.homma.c8@tohoku.ac.jp

[2] Graduate School of Informatics, Kyoto University, Yoshida-honmachi, Sakyo-ku, Kyoto, 606-8501, Japan, ueno.rei.2e@kyoto-u.ac.jp

Abstract. This paper proposes a high-speed hardware accelerator for QR-UOV, a multivariate scheme, that executes all three operations: key generation, signature generation, and signature verification. QR-UOV utilizes a quotient polynomial ring structure to reduce the public-key size of the original UOV scheme; however, this introduces functional requirements distinct from other multivariate schemes, such as polynomial-matrix operations over $\mathbb{F}_{q^\ell}$, coefficient expansion for the Mersenne prime field $\mathbb{F}_q$, and the expansion of symmetric matrices generated in a compressed form. To accelerate polynomial-matrix multiplication over $\mathbb{F}_{q^\ell}$, the proposed architecture employs a $D \times D$ systolic array (D is the parallelization factor), in which each PE internally contains a polynomial multiply-accumulate unit. For signature verification, the architecture also incorporates a quadratic-form operation unit without explicitly storing the intermediate vector–matrix products. In addition, to ensure a regular data supply from key expansion to matrix operations, the architecture incorporates on-the-fly rejection sampling, which exploits the property that AES-CTR output blocks can be generated independently, as well as an expansion process for the symmetric matrices in compressed form. Furthermore, the architecture enables pipelined processing, overlapping the i-th polynomial-matrix operation with the $(i+1)$-th public-key expansion. FPGA evaluation shows that the proposed hardware executes key generation, signature generation, and signature verification in 0.74 ms, 0.28 ms, and 0.19 ms, respectively, at security level I. Furthermore, we confirm that the proposed hardware exhibits a lower LUT-time product than that of existing UOV hardware.

Keywords: post-quantum cryptography · hardware accelerator · multivariate signatures · QR-UOV

1 Introduction

1.1 Background

In 2022, the National Institute of Standards and Technology (NIST) initiated a call for additional digital signature proposals to be considered in the Post-Quantum Cryptography (PQC) standardization process. QR-UOV (Quotient Ring Unbalanced Oil and Vinegar) [FIKT21] is one of the Round 3 multivariate schemes, announced in 2026 [MAB+26]. It reduces the public-key size of the original Unbalanced Oil and Vinegar (UOV) [KPG99], which is also a Round 3 scheme, by incorporating a quotient polynomial ring structure. On the other hand, QR-UOV poses implementation challenges in hardware that differ from those of other multivariate schemes such as UOV [BCD+24] and MAYO [BCC+24].

In particular, it requires polynomial-matrix operations over $\mathbb{F}_{q^\ell}$, rejection sampling for the Mersenne prime field $\mathbb{F}_q$, and the expansion of symmetric matrices generated in a compressed form. Furthermore, the public-key matrix is expanded from a seed on a per-iteration basis and the expanded matrix is fed immediately into the subsequent polynomial-matrix operation. Due to these structural characteristics, a hardware architecture tailored to the unique data-generation order, matrix structure, and polynomial operations specific to QR-UOV should be considered separately. Nevertheless, to date, no hardware architecture covering all QR-UOV operations, including key generation, signature generation, and signature verification, has been reported (while a hardware implementation for limited sub-operations, such as rejection sampling, exists [IKA+25]). Although some optimized UOV and MAYO software/hardware implementations and an optimized QR-UOV software implementation have been developed (see Section 2.3), a research question—how we can accelerate QR-UOV on hardware—arises.

1.2 Our contributions

This paper proposes a hardware accelerator that covers all QR-UOV operations and enables their high-speed execution. The main features of the proposed hardware are as follows:

- **Parallelized polynomial-matrix operations over $\mathbb{F}_{q^\ell}$:** The proposed architecture employs a $D \times D$ systolic array (D is the parallelization factor), in which each Processing Element (PE) performs polynomial multiply-accumulate operations over $\mathbb{F}_{q^\ell}$, thereby accelerating the polynomial-matrix multiplications required in KeyGen and Sign. For Verify, it incorporates a dedicated quadratic-form operation unit that directly accumulates products using the polynomial-block representation, without storing intermediate vector–matrix multiplication results.
- **Accelerated coefficient and symmetric-matrix expansion:** The datapath incorporates on-the-fly rejection sampling, which exploits the independence of the output blocks of the pseudorandom generator (i.e., AES-CTR) to generate $\mathbb{F}_q$ coefficients without intermediate memory storage, and block-matrix-wise symmetric matrix expansion to supply data regularly to the subsequent polynomial-matrix operation units.
- **Parallel execution of public-key expansion and polynomial-matrix operations:** The datapath executes the i-th polynomial-matrix operation and the $(i+1)$-th public-key expansion in parallel during key generation, signature generation, and signature verification. Unlike UOV and MAYO, QR-UOV independently expands the public-key matrix for each coefficient of the quotient polynomial, enabling pipeline processing across iterations of the loop over the public-key components.

Summary of results This paper presents the evaluation results of the proposed hardware on an Artix-7 FPGA, which is commonly used as a benchmarking hardware platform for PQC schemes. In particular, we demonstrate that the proposed hardware executes key generation, signature generation, and signature verification in 0.74 ms, 0.28 ms, and 0.19 ms, respectively, at security level I. Furthermore, compared with existing UOV implementations, we show that the proposed hardware achieves a lower LUT-time product. These results demonstrate that our hardware can efficiently execute the entire QR-UOV processing on an FPGA and exhibits high efficiency as a hardware accelerator for multivariate schemes.

1.3 Paper organization

Section 2 briefly introduces QR-UOV and the relevant background knowledge. Section 3 describes the architecture and scheduling of the proposed hardware. Section 4 presents

the implementation results of the proposed hardware and provides comparisons with other multivariate schemes. Section 5 discusses the proposed hardware in further detail. Finally, Section 6 concludes the paper.

2 Preliminaries

2.1 Overview of QR-UOV

Multivariate polynomial cryptography is a family of cryptographic schemes whose security is based on the multivariate quadratic (MQ) problem, i.e., the hardness of solving systems of quadratic equations over finite fields. Since QR-UOV is a derivative of UOV (Unbalanced Oil and Vinegar) [KPG99], a foundational multivariate polynomial signature scheme, we first describe the UOV signature scheme.

Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be a vector of variables over the finite field $\mathbb{F}_q$, where $x_1, \ldots, x_v$ are called vinegar variables and $x_{v+1}, \ldots, x_n$ are called oil variables. Let the central map be $\mathcal{F} = (f_1, \ldots, f_m) : \mathbb{F}_q^n \to \mathbb{F}_q^m$, where each $f_k,\ k \in [m]$ is given by the following polynomial with coefficients $\alpha_{i,j}^{(k)} \in \mathbb{F}_q$:

$$f_k(x_1, \ldots, x_n) = \sum_{i=1}^{v} \sum_{j=i}^{n} \alpha_{i,j}^{(k)} x_i x_j. \tag{1}$$

An invertible linear map $\mathcal{S} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ is then chosen at random to conceal the structure of $\mathcal{F}$. The public-key map $\mathcal{P}$ is given as the polynomial map

$$\mathcal{P} = \mathcal{F} \circ \mathcal{S} : \mathbb{F}_q^n \to \mathbb{F}_q^m,$$

where $\mathcal{F}$ and $\mathcal{S}$ serve as the secret key.

We next describe how to compute the inverse of the central map $\mathcal{F}$. Given $\boldsymbol{t} \in \mathbb{F}_q^m$, we seek $\boldsymbol{x} \in \mathbb{F}_q^n$ satisfying $\mathcal{F}(\boldsymbol{x}) = \boldsymbol{t}$. First, values $y_1, \ldots, y_v$ over $\mathbb{F}_q$ are randomly chosen as the vinegar variables. The solution is then obtained by solving the equation $\mathcal{F}(y_1, \ldots, y_v, x_{v+1}, \ldots, x_n) = \boldsymbol{t}$. Since the central map $\mathcal{F}$ is defined as in Eq. (1), this equation becomes a system of m linear equations in $n - v$ oil variables. If no solution exists, new vinegar variables $y_1', \ldots, y_v'$ are resampled, and this process is repeated until a solution is found.

Signature generation proceeds as follows, based on the above inversion procedure. Given a message to be signed as $\boldsymbol{t} \in \mathbb{F}_q^m$, we find a solution $\boldsymbol{y}$ satisfying $\mathcal{F}(\boldsymbol{y}) = \boldsymbol{t}$, and the signature $\boldsymbol{s}$ is then given by

$$\boldsymbol{s} = \mathcal{S}^{-1}(\boldsymbol{y}) \in \mathbb{F}_q^n.$$

Finally, we describe how the public and secret keys are represented in UOV. For each polynomial p_i constituting the public-key map $\mathcal{P}$, there exists an $n \times n$ matrix P_i such that $p_i(\boldsymbol{x}) = \boldsymbol{x}^\top \cdot P_i \cdot \boldsymbol{x}$. Such matrices $P_1, \ldots, P_m$ are referred to as the public-key matrices. Similarly, an $n \times n$ matrix F_i is associated with each f_i for $i \in [m]$, and the $n \times n$ matrix S is defined by $\mathcal{S}(\boldsymbol{x}) = S \cdot \boldsymbol{x}$. In general, the matrices P_i and F_i are given as symmetric matrices when q is odd, and as upper triangular matrices when q is even. Since F_i must satisfy Eq. (1), it is given as follows.

$$\begin{pmatrix} *_{v \times v} & *_{v \times m} \\ *_{m \times v} & 0_{m \times m} \end{pmatrix}.$$

Here, with $\mathcal{P} = \mathcal{F} \circ \mathcal{S}$, P_i is given as

$$P_i = S^\top F_i S. \tag{2}$$

Algorithm 1 KeyGen() [AFI+24]

Ensure: public key $\mathsf{pk} \in \{0,1\}^{\lambda} \times \left(\mathbb{F}_{q^\ell}^{M\times M}\right)^m$ and private key $\mathsf{sk} \in \{0,1\}^{2\lambda}$

1: $(\mathsf{seed}_{\mathsf{pk}}, \mathsf{seed}_{\mathsf{sk}}) \xleftarrow{\$} \{0,1\}^{2\lambda}$ ▷ $\mathsf{seed}_{\mathsf{pk}}, \mathsf{seed}_{\mathsf{sk}} \in \{0,1\}^{\lambda}$
2: $\bar{S}' \leftarrow \mathsf{Expand}_{\mathsf{sk}}(\mathsf{seed}_{\mathsf{sk}})$ ▷ $\bar{S}' \in \mathbb{F}_{q^\ell}^{V\times M}$
3: **for** i from 1 to m **do**
4: $\quad(\bar{P}_{i,1}, \bar{P}_{i,2}) \leftarrow \mathsf{Expand}_{\mathsf{pk}}(\mathsf{seed}_{\mathsf{pk}}, i)$ ▷ $\bar{P}_{i,1} \in \mathbb{F}_{q^\ell}^{V\times V}$ (symmetric), $\bar{P}_{i,2} \in \mathbb{F}_{q^\ell}^{V\times M}$
5: $\quad\bar{P}_{i,3} \leftarrow -\bar{S}'^{\top}\bar{P}_{i,1}\bar{S}' + \bar{P}_{i,2}^{\top}\bar{S}' + \bar{S}'^{\top}\bar{P}_{i,2}$ ▷ $\bar{P}_{i,3} \in \mathbb{F}_{q^\ell}^{M\times M}$
6: **end for**
7: **return** $(\mathsf{pk}, \mathsf{sk}) = ((\mathsf{seed}_{\mathsf{pk}}, \{\bar{P}_{i,3}\}_{i\in[m]}), (\mathsf{seed}_{\mathsf{sk}}, \mathsf{seed}_{\mathsf{pk}}))$

The above descriptions are common to both QR-UOV and UOV.

We then describe the polynomial-matrix representation, whose elements are given by the quotient polynomial ring, which distinguishes QR-UOV from UOV. Let ℓ be a positive integer and let $f \in \mathbb{F}_q[x]$ be an irreducible polynomial satisfying $\deg f = \ell$. For any element g of the quotient polynomial ring $\mathbb{F}_q[x]/(f)$, one can uniquely define an $\ell \times \ell$ matrix Φ_g^f over $\mathbb{F}_q$ satisfying the following:

$$(1 \;\; x \;\; \cdots \;\; x^{\ell-1})\Phi_g^f = (g \;\; xg \;\; \cdots \;\; x^{\ell-1}g).$$

The map $g \mapsto \Phi_g^f$ is an injective ring homomorphism, and for any $g_1, g_2 \in \mathbb{F}_q[x]/(f)$, the following relations hold:

$$\Phi_{g_1+g_2}^f = \Phi_{g_1}^f + \Phi_{g_2}^f \quad \text{and} \quad \Phi_{g_1 g_2}^f = \Phi_{g_1}^f \Phi_{g_2}^f.$$

Here, the matrix algebra $\mathcal{A}_f$ is defined as

$$\mathcal{A}_f := \left\{\Phi_g^f \in \mathbb{F}_q^{\ell\times\ell} \mid g \in \mathbb{F}_q[x]/(f)\right\}.$$

Since Φ_g^f is determined by the ℓ coefficients of $g \in \mathbb{F}_q[x]/(f)$, any $\ell \times \ell$ polynomial matrix Φ_g^f in $\mathcal{A}_f$ can be represented by ℓ elements over $\mathbb{F}_q$. QR-UOV exploits this property to compress the public-key size compared to UOV.

2.2 QR-UOV algorithm

In this subsection, we describe the key generation (KeyGen), signature generation (Sign), and signature verification (Verify) procedures of QR-UOV as specified in the Round 2 submission [AFI+24]. We also describe the seed expansion procedure and the parameter sets that are common to all three operations.

Key generation Algorithm 1 shows the KeyGen algorithm of QR-UOV, where the public key pk and the secret key sk are generated from two seed values $\mathsf{seed}_{\mathsf{sk}}$ and $\mathsf{seed}_{\mathsf{pk}}$. Note here that the $\mathbb{F}_{q^\ell}$ matrix generated from the seed is computed without expanding it into an $\mathbb{F}_q$ matrix. We define $\mathbb{F}_{q^\ell} = \mathbb{F}_q[x]/(f)$ using an irreducible polynomial $f \in \mathbb{F}_q[x]$ of degree ℓ, and let Φ_g^f denote the polynomial matrix determined by an arbitrary element $g \in \mathbb{F}_{q^\ell}$. We also define the matrix $\mathcal{A}_f := \left\{\Phi_g^f \mid g \in \mathbb{F}_{q^\ell}\right\}$ and choose $W \in \mathbb{F}_q^{\ell\times\ell}$ such that $W\mathcal{A}_f$ consists of symmetric matrices. The block diagonal matrix with a copies of W arranged along the diagonal is denoted $W^{(a)} \in \mathbb{F}_q^{a\ell\times a\ell}$. Let v denote the number of vinegar variables and m the number of oil variables, and define V, M, and N as follows:

$$v = \ell V, \quad m = \ell M, \quad N := V + M.$$

Let $\mathcal{A}_f^{a\times b}$ denote an $a \times b$ block matrix whose entries belong to $\mathcal{A}_f$. Then, the public-key matrices P_i ($i \in [m]$) and the central map F_i are elements of $W^{(a)}\mathcal{A}_f^{N\times N}$, whereas the matrix S representing the linear map $\mathcal{S}$ that constitutes the secret key belongs to $\mathcal{A}_f^{N\times N}$. Since each block in $\mathcal{A}_f$ is an $\ell \times \ell$ matrix, we have $P_i, F_i, S \in \mathbb{F}_q^{n\times n}$.

For each i, the matrices P_i, F_i, and S are expressed as the following block matrices.

$$P_i = \begin{pmatrix} P_{i,1} & P_{i,2} \\ P_{i,2}^\top & P_{i,3} \end{pmatrix},\ F_i = \begin{pmatrix} F_{i,1} & F_{i,2} \\ F_{i,2}^\top & 0_{m\times m} \end{pmatrix},\ S = \begin{pmatrix} I_v & S' \\ 0_{m\times v} & I_m \end{pmatrix},$$

where $P_{i,1}, F_{i,1} \in W^{(V)}\mathcal{A}_f^{V\times V},\ P_{i,2}, F_{i,2} \in W^{(V)}\mathcal{A}_f^{V\times M}, P_{i,3} \in W^{(M)}\mathcal{A}_f^{M\times M}, S' \in \mathcal{A}_f^{V\times M}$. Since P_i and F_i are symmetric matrices, $P_{i,1}$ and $F_{i,1}$ are $v \times v$ symmetric matrices, and $P_{i,3}$ is an $m \times m$ symmetric matrix. According to Eq. (2),

$$\begin{aligned} F_{i,1} &= P_{i,1}, \\ F_{i,2} &= -P_{i,1}S' + P_{i,2}, \\ P_{i,3} &= -S'^\top P_{i,1}S' + P_{i,2}^\top S' + S'^\top P_{i,2}. \end{aligned}$$

Since $P_{i,3}$ can be computed from $P_{i,1}$, $P_{i,2}$, and S', the matrices $P_{i,1}$, $P_{i,2}$, and S' are expanded from seeds, and $P_{i,3}$ is included in the public key. The public key pk and the secret key sk are therefore given by

$$\begin{aligned} \mathsf{pk} &= (\mathsf{seed}_{\mathsf{pk}},\ \{P_{i,3}\}_{i\in[m]}), \\ \mathsf{sk} &= (\mathsf{seed}_{\mathsf{sk}},\ \mathsf{seed}_{\mathsf{pk}}). \end{aligned}$$

Here, $P_{i,1}$ and $P_{i,2}$ are expanded using $\mathsf{seed}_{\mathsf{pk}}$, and S' is expanded using $\mathsf{seed}_{\mathsf{sk}}$.

For efficient implementation, QR-UOV does not compute directly using $P_{i,1}$, $P_{i,2}$, and S', but instead performs computations using $\bar{P}_i$, $\bar{F}_i$, and $\bar{S}'$ over $\mathbb{F}_{q^\ell}$. Defining the map ϕ as

$$\phi : \mathcal{A}_f^{a\times b} \mapsto \mathbb{F}_{q^\ell}^{a\times b}, \quad (\Phi_{g_{i,j}}^f) \mapsto (g_{i,j})_{i,j},$$

the following relations hold:

$$\bar{P}_i = \phi\Big((W^{(N)})^{-1}P_i\Big), \quad \bar{F}_i = \phi\Big((W^{(N)})^{-1}F_i\Big), \quad \bar{S} = \phi(S).$$

Under these definitions, the relation (2) among P_i, F_i, and S corresponds to $\bar{P}_i = \bar{S}^\top \bar{F}_i \bar{S}$ over $\mathbb{F}_{q^\ell}$.

Algorithm 1 is expressed with computations over $\mathbb{F}_{q^\ell}$; it expands $\bar{P}_{i,1}$ and $\bar{P}_{i,2}$ from $\mathsf{seed}_{\mathsf{pk}}$, expands $\bar{S}'$ from $\mathsf{seed}_{\mathsf{sk}}$, and computes $P_{i,3}$ in compressed form via the relation

$$\bar{P}_{i,3} = -\bar{S}'^\top \bar{P}_{i,1}\bar{S}' + \bar{P}_{i,2}^\top \bar{S}' + \bar{S}'^\top \bar{P}_{i,2}.$$

Signature generation Algorithm 2 shows the Sign algorithm, where a message $\boldsymbol{M}$ and the secret key sk are given as inputs and a signature σ is generated. This procedure follows the same basic structure as UOV.

First, the vinegar variables $y_1, \ldots, y_v \in \mathbb{F}_q$ are chosen uniformly at random. Then, a λ-bit salt r and

$$\mu = \mathsf{SHAKE256}(\mathsf{seed}_{\mathsf{pk}} \| \boldsymbol{M})$$

are generated, from which $\boldsymbol{t} \coloneqq \mathsf{Hash}(\mu, r) \in \mathbb{F}_q^m$ is computed. Here, Hash applies SHAKE256 followed by the rejection sampling procedure. This yields a system of linear equations in the oil variables $x_{v+1}, \ldots, x_n \in \mathbb{F}_q$:

$$\mathcal{F}(y_1, \ldots, y_v, x_{v+1}, \ldots, x_n) = \boldsymbol{t}. \tag{3}$$

Algorithm 2 $\mathsf{Sign}(\boldsymbol{M}, \mathsf{sk})$ [AFI+24]

Require: message $\boldsymbol{M} \in \mathbb{B}^*$ and private key $\mathsf{sk} \in \{0,1\}^{2\lambda}$
Ensure: signature $\sigma \in \{0,1\}^{\lambda} \times \mathbb{F}_q^n$
1: $(\mathsf{seed}_{\mathsf{sk}}, \mathsf{seed}_{\mathsf{pk}}) \leftarrow \mathsf{sk}$
2: $\bar{S}' \leftarrow \mathsf{Expand}_{\mathsf{sk}}(\mathsf{seed}_{\mathsf{sk}})$
3: $\boldsymbol{y} = (y_1, \ldots, y_v)^\top \xleftarrow{\$} \mathbb{F}_q^v$
4: **for** i from 1 to m **do**
5: $\quad (\bar{P}_{i,1}, \bar{P}_{i,2}) \leftarrow \mathsf{Expand}_{\mathsf{pk}}(\mathsf{seed}_{\mathsf{pk}}, i)$
6: $\quad \bar{\boldsymbol{y}} \leftarrow W^{(V)}\phi^{-1}(\bar{P}_{i,1})\boldsymbol{y}$ ▷ $\bar{\boldsymbol{y}} \in \mathbb{F}_q^v$
7: $\quad \boldsymbol{L}_i \leftarrow -2\phi^{-1}(\bar{S}')^\top \bar{\boldsymbol{y}} + 2W^{(M)}\phi^{-1}(\bar{P}_{i,2}^\top)\boldsymbol{y}$ ▷ $\boldsymbol{L}_i \in \mathbb{F}_q^m$
8: $\quad u_i \leftarrow \boldsymbol{y}^\top \bar{\boldsymbol{y}}$ ▷ $u_i \in \mathbb{F}_q$
9: **end for**
10: $L \leftarrow (\boldsymbol{L}_1, \ldots, \boldsymbol{L}_m)^\top$ ▷ $L \in \mathbb{F}_q^{m \times m}$
11: $\boldsymbol{u} \leftarrow (u_1, \ldots, u_m)^\top$ ▷ $\boldsymbol{u} \in \mathbb{F}_q^m$
12: $\mu \leftarrow \mathsf{SHAKE256}(\mathsf{seed}_{\mathsf{pk}} || \mathsf{BytesToBits}(\boldsymbol{M}), 512)$
13: **repeat**
14: $\quad r \xleftarrow{\$} \{0,1\}^{\lambda}$
15: $\quad \boldsymbol{t} \leftarrow \mathsf{Hash}(\mu, r)$ ▷ $\boldsymbol{t} \in \mathbb{F}_q^m$
16: **until** $L\boldsymbol{x} = \boldsymbol{t} - \boldsymbol{u}$ has solutions for $\boldsymbol{x}$.
17: Choose one solution $(y_{v+1}, \ldots, y_n) \in \mathbb{F}_q^m$ of $L\boldsymbol{x} = \boldsymbol{t} - \boldsymbol{u}$ randomly.
18: $\boldsymbol{s} \leftarrow (y_1, \ldots, y_v, y_{v+1}, \ldots, y_n)^\top - \left(\phi^{-1}(\bar{S}') \cdot (y_{v+1}, \ldots, y_n)^\top || \boldsymbol{0}_m\right)$ ▷ $\boldsymbol{s} \in \mathbb{F}_q^n$
19: **return** $\sigma = (r, \boldsymbol{s})$

If Eq. (3) has at least one solution, values $y_{v+1}, \ldots, y_n$ satisfying Eq. (3) are chosen uniformly at random, and the signature σ is generated by applying $\mathcal{S}^{-1}$ to $(y_1, \ldots, y_n)$. If Eq. (3) has no solution, a new salt r is generated and $\boldsymbol{t}$ is updated accordingly, repeating this process until Eq. (3) admits a solution. A key difference from UOV Sign is that, when Eq. (3) has no solution, the salt r is resampled rather than the vinegar variables. This ensures that the signature $\boldsymbol{s}$ satisfying $\mathcal{P}(\boldsymbol{s}) = \boldsymbol{t}$ is uniformly distributed over $\mathbb{F}_q^n$.

Signature verification Algorithm 3 shows the Verify algorithm, where a message $\boldsymbol{M}$, the public key pk, and a signature σ are inputs, and either accept or reject is output. The verification procedure follows the same flow as the UOV verification. First, as in Sign, μ and $\boldsymbol{t}$ are computed from $\mathsf{seed}_{\mathsf{pk}}$, $\boldsymbol{M}$, and r. Next, the public-key map $\mathcal{P}$ is applied to the signature $\boldsymbol{s} \in \mathbb{F}_q^n$ to compute $\boldsymbol{t}' \coloneqq \mathcal{P}(\boldsymbol{s}) \in \mathbb{F}_q^m$. The signature is accepted if and only if $\boldsymbol{t} = \boldsymbol{t}'$; otherwise, it is rejected.

Seed expansion In QR-UOV, the public-key matrices $\bar{P}_{i,1}$, $\bar{P}_{i,2}$ and the secret key matrix $\bar{S}'$ are expanded from seeds using a pseudorandom generator (PRG). The PRG output is mapped to uniformly random values over $\mathbb{F}_q$ via rejection sampling (RejSamp), yielding the coefficients of polynomials over $\mathbb{F}_{q^\ell}$. RejSamp operates on a byte-by-byte basis, taking as input a byte string $(r_1, \ldots, r_\tau) \in \mathbb{B}^\tau$, the byte length τ, and the desired output vector length n', and producing $(v_1, \ldots, v_{n'}) \in \mathbb{F}_q^{n'}$. Here, the value of τ is specified by the QR-UOV specification and satisfies $\tau > n'$. Each byte of the input string is bitwise ANDed with q to produce a vector with entries in $\{0, \ldots, q\}$. The resulting vector is then partitioned into two parts: the "head" sequence consisting of the first n' bytes, and the "tail" sequence consisting of the remaining $\tau - n'$ bytes. The head sequence is then inspected, and any entry equal to q is replaced by the first non-q entry found in the tail sequence. After this process, the head sequence constitutes a vector over $\mathbb{F}_q$. The value of

Algorithm 3 $\mathsf{Verify}(\boldsymbol{M}, \mathsf{pk}, \sigma)$ [AFI+24]

Require: message $\boldsymbol{M} \in \mathbb{B}^*$, public key $\mathsf{pk} \in \{0,1\}^\lambda \times \left(\mathbb{F}_{q^\ell}^{M \times M}\right)^m$, and signature $\sigma \in \{0,1\}^\lambda \times \mathbb{F}_q^n$
Ensure: accept or reject
1: $(\mathsf{seed}_{\mathsf{pk}}, \{\bar{P}_{i,3}\}_{i\in[m]}) \leftarrow \mathsf{pk}$
2: $(r, \boldsymbol{s}) \leftarrow \sigma$
3: **for** i from 1 to m **do**
4: $\quad(\bar{P}_{i,1}, \bar{P}_{i,2}) \leftarrow \mathsf{Expand}_{\mathsf{pk}}(\mathsf{seed}_{\mathsf{pk}}, i)$
5: $\quad P_i \leftarrow W^{(N)}\phi^{-1}\begin{pmatrix} \bar{P}_{i,1} & \bar{P}_{i,2} \\ \bar{P}_{i,2}^\top & \bar{P}_{i,3} \end{pmatrix}$ ▷ $P_i \in \mathbb{F}_q^{n\times n}$
6: **end for**
7: $\mu \leftarrow \mathsf{SHAKE256}(\mathsf{seed}_{\mathsf{pk}} || \mathsf{BytesToBits}(\boldsymbol{M}), 512)$
8: $\boldsymbol{t} \leftarrow \mathsf{Hash}(\mu, r)$
9: $\boldsymbol{t'} \leftarrow (\boldsymbol{s}^\top P_1 \boldsymbol{s}, \ldots, \boldsymbol{s}^\top P_m \boldsymbol{s})^\top$ ▷ $\boldsymbol{t'} \in \mathbb{F}_q^m$
10: **return** accept if $\boldsymbol{t} = \boldsymbol{t'}$ and reject otherwise.

Table 1: QR-UOV Round 2 recommended parameter sets [AFI+24]

SL	(q, v, m, ℓ)	n	N	V	M	f
I	(127, 156, 54, 3)	210	70	52	18	
III	(127, 228, 78, 3)	306	102	76	26	$x^3 - x - 1$
V	(127, 306, 105, 3)	411	137	102	35	

τ is chosen to be sufficiently large so that the non-q entries in the tail sequence are enough to replace all occurrences of q in the head sequence.

In Algorithms 1, 2, and 3, the procedures that require RejSamp are $\mathsf{Expand}_{\mathsf{sk}}$, $\mathsf{Expand}_{\mathsf{pk}}$, and Hash, where $\mathsf{Expand}_{\mathsf{sk}}$ and $\mathsf{Expand}_{\mathsf{pk}}$ convert PRG output into vectors over $\mathbb{F}_q$, and Hash converts SHAKE256 output into vectors over $\mathbb{F}_q$. In the expansion of the secret key matrix $\bar{S}' \in \mathbb{F}_{q^\ell}^{V\times M}$ by $\mathsf{Expand}_{\mathsf{sk}}$, polynomials over $\mathbb{F}_{q^\ell}$ are generated in ascending order of degree from ℓVM elements over $\mathbb{F}_q$. Similarly, the public-key matrices $\bar{P}_{i,1} \in \mathbb{F}_{q^\ell}^{V\times V}$ and $\bar{P}_{i,2} \in \mathbb{F}_{q^\ell}^{V\times M}$ are expanded in the same manner. When expanding the symmetric matrix $\bar{P}_{i,1} \in \mathbb{F}_{q^\ell}^{V\times V}$, only the upper triangular entries including the diagonal are generated, so the number of required elements over $\mathbb{F}_q$ is $\ell V(V+1)/2$.

Parameters Table 1 lists the recommended Round 2 parameter sets with $(q, \ell) = (127, 3)$ for security levels I, III, and V (SL-I, SL-III, and SL-V), respectively. The security parameter λ, which determines the sizes of $\mathsf{seed}_{\mathsf{pk}}$, $\mathsf{seed}_{\mathsf{sk}}$, and the salt r, is set to 128, 192, and 256 for SL-I, SL-III, and SL-V, respectively. See the QR-UOV specification [AFI+24] for the other parameter sets.

2.3 Related work

Several implementations of multivariate schemes have been reported, covering UOV, MAYO, and Rainbow. Beullens et al. [BCH+23] implemented UOV on multiple platforms, including an x86 processor, Arm Neon, Cortex-M4, and FPGA. Their work exploits the characteristics of each platform, and demonstrates an execution with minimal hardware resource usage on FPGA. For MAYO, Hirner et al. [HSK+24] reported a high-speed FPGA and ASIC accelerator. Their design incorporates partial loop unrolling for the matrix element generation loop, memory structure optimization, and a reduction in on-chip

memory usage by feeding PRG output directly into the arithmetic units without routing it through memory. In addition, [SMA+24] reported a MAYO implementation targeting resource-constrained devices, with careful consideration of the trade-off between execution time and circuit area. For Rainbow, [FG18] reported a high-speed FPGA implementation targeting the NIST Round 1 version of Rainbow. This implementation focuses primarily on Sign and presents an architecture for efficiently solving systems of linear equations over finite fields.

However, since UOV, MAYO, and Rainbow operate over even-characteristic fields such as $\mathrm{GF}(2^4)$ and $\mathrm{GF}(2^8)$, their implementations differ substantially from those of QR-UOV, which operates over odd-characteristic fields with $q = 7$, 31, and 127. First, the PRG output cannot be used directly in QR-UOV, and a conversion process is required to map the output to coefficients over the finite field. Also, the order in which coefficients are generated differs between QR-UOV and other schemes, resulting in different memory mapping. Moreover, since the public key and secret key matrices are computed as polynomial matrices, the arithmetic optimization strategies also differ. Therefore, while some techniques from existing implementations are applicable to QR-UOV, such as the linear system solver of [BCH+23], it is generally difficult to directly apply the efficient implementation techniques of other multivariate schemes to QR-UOV, and a dedicated architecture tailored to its coefficient generation procedure and matrix structure is required[1].

A few implementation techniques targeting QR-UOV have been reported to date. Amagasa et al. [AUH26] reported a fast software implementation of QR-UOV, presenting techniques optimized for modern x86 processors. Their work accelerates the primary computation of QR-UOV—matrix operations over $\mathbb{F}_{q^\ell}$—using SIMD instructions. They also present a technique for accelerating RejSamp, which is specific to QR-UOV owing to its use of $q = 7$, 31, and 127; this technique is also applicable to hardware implementations. Furthermore, their work is the first to accelerate the full processing pipeline of QR-UOV, and through bottleneck analysis, identifies polynomial-matrix multiplication in KeyGen and public-key expansion in Verify as the most time-consuming operations.

On the other hand, a hardware architecture for RejSamp applicable to QR-UOV, known as the RejSCore architecture, has been reported [IKA+25]. This architecture performs RejSamp iteratively against an AES-CTR-based PRG, minimizing circuit area at the cost of clock cycle count (i.e., throughput). Consequently, a high-speed hardware implementation remains unexplored. Furthermore, since [IKA+25] targets only RejSamp and does not implement all QR-UOV operations—encompassing KeyGen, Sign, and Verify—the overall effectiveness of a complete hardware implementation has remained unclear.

3 Proposed method

This section describes the proposed method for efficiently executing QR-UOV KeyGen, Sign, and Verify in hardware. The proposed architecture targets the Round 2 QR-UOV specification [AFI+24] and adopts the option that uses AES-CTR as the PRG[2] for high-speed execution.

3.1 Overall structure

Figure 1 shows a block diagram of the overall structure of the proposed hardware, which consists of a seed expansion unit, a polynomial-matrix operation unit, a linear equation solver unit, and internal memory. The seed expansion unit generates $\mathbb{F}_q$ coefficients from seeds and expands compressed symmetric matrices into a format suitable for the

[1] Although [SMA+24] also presents an implementation for $q = 31$, the conversion process to finite field coefficients is not addressed, making direct application to QR-UOV impractical.

[2] AES-128, AES-192, and AES-256 are used for SL-I, SL-III, and SL-V, respectively.

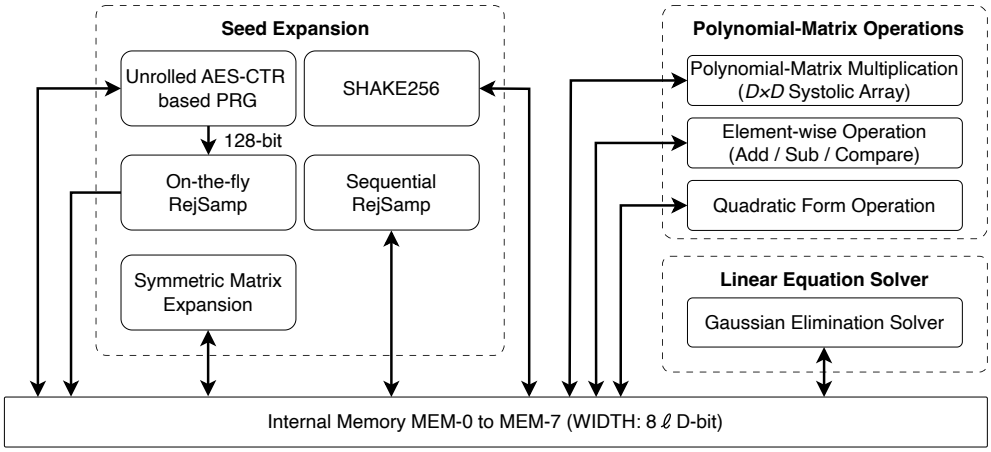


Figure 1: Overview of proposed hardware.

subsequent matrix operations. The polynomial-matrix operation unit parallelizes the dominant polynomial-matrix multiplication required in KeyGen and Sign using a $D \times D$ systolic array in which each PE internally contains a polynomial multiply-accumulate unit over $\mathbb{F}_{q^\ell}$, while additions, subtractions, and comparisons are performed element-wise. For Verify, the quadratic-form operation unit directly computes each component of the public map without storing intermediate vector–matrix multiplication results. The linear equation solver unit solves the system of linear equations that arises during Sign using Gaussian elimination. These units are interconnected via internal memory that provides regular data supply between the public-key expansion and the matrix operations during the processing loop over the public-key components $\bar{P}_i$ $(i = 1, \ldots, m)$. The details of the memory configuration are described in Section 3.5. By varying V and M, the proposed architecture can accommodate the parameter sets for SL-I, SL-III, and SL-V.

3.2 Seed expansion

The seed expansion unit expands the secret-key and public-key matrices using a PRG, and also generates coefficient sequences over $\mathbb{F}_q$ from SHAKE256 output. In the proposed architecture, AES-CTR is used for the PRG. It was reported in [AUH26] that a primary bottleneck of QR-UOV software is the pseudorandom number generation (i.e., AES) during Verify. To address this, the proposed architecture adopts a high-throughput, fully unrolled implementation of AES-CTR [BCH+23]. Specifically, based on the structure of [Hsi12], all rounds are unrolled so that one ciphertext block is generated per clock cycle, with pipeline registers inserted between rounds. On the other hand, SHAKE256 is implemented based on the design reported in [RB20], which has a datapath for one round function and executes 24 rounds at a rate of one round per clock cycle since the throughput is not critical for the overall process. In either case, RejSamp is required to generate $\mathbb{F}_q$ elements from the corresponding pseudorandom sequence as described below.

Rejection sampling (RejSamp) In QR-UOV, since n-bit values obtained from the PRG or SHAKE256 take values in $[0, 2^n)$, RejSamp is applied to convert the pseudorandom sequence into a sequence of elements of the Mersenne prime field $\mathbb{F}_q$ with $q = 2^n - 1$. Specifically, RejSamp partitions the pseudorandom sequence into a head portion and a tail portion. Whenever the value q appears in the head portion, it is replaced by a valid non-q element drawn from the tail portion. As prior work on RejSamp for QR-UOV, [IKA+25]

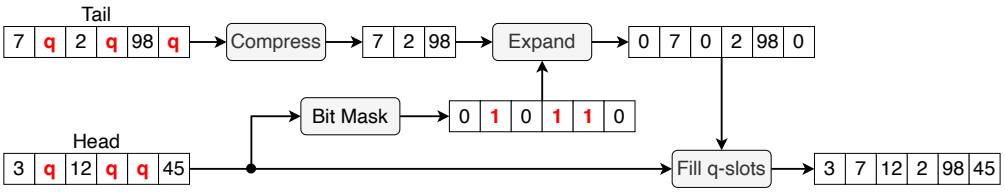


Figure 2: RejSamp flow.

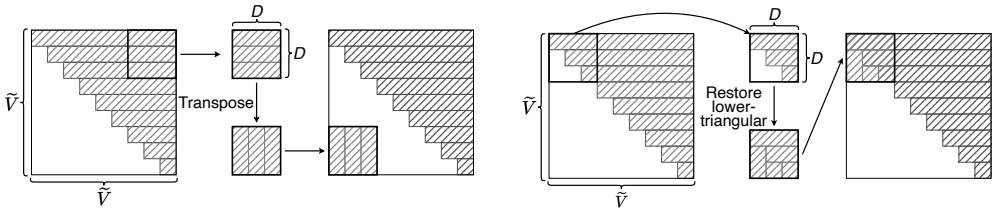
presents an architecture that minimizes circuit area at the cost of clock cycle count. In contrast, for high-speed execution, we introduce a high-throughput RejSamp method based on the approach presented in [AUH26].

Figure 2 illustrates the flow of the proposed RejSamp. First, for the tail portion, only valid $\mathbb{F}_q$ elements excluding q are packed into a buffer as a dense vector. Then, for each position in the head portion where the value q appears, the corresponding entry is replaced with an element from the dense vector in the buffer. The data to be packed and the positions at which the dense vector elements are placed are controlled by mask bits. A mask that selects valid non-q values is used to compress the tail portion, while another mask indicating the positions where q appears is used to expand the head portion. Note here that the compression and expansion operations in QR-UOV software [AUH26] were implemented with SIMD instructions, which are not directly used in hardware.

In the proposed hardware, we introduce the bit-compression pex and bit-expansion pdep operations of Hilewitz and Lee [HL06] to implement byte-level compression and expansion operations. In [HL06], these operations are implemented by placing mask-based bit swaps at each stage of a butterfly network applied to the input bit string. As an example, 64-bit compression and expansion can be implemented by pipelining a six-stage butterfly network with bit swaps of 1, 2, 4, 8, 16, and 32 bits. In the proposed method, we extend the approach to byte-wise operations since $\mathbb{F}_q$ elements are handled at the byte level. As a result, the proposed RejSamp for AES-CTR and SHAKE256 can process 16 and ℓD elements in parallel per clock cycle, respectively, whereas the existing RejSCore architecture [IKA+25] processes only one element per clock cycle.

On-the-fly AES-CTR-based RejSamp The proposed RejSamp architecture realizes on-the-fly processing that immediately samples the generated pseudorandom sequence as elements of $\mathbb{F}_q$. In the conventional approach, the head sequence is first output from the PRG and temporarily stored in a memory buffer, which must then be read back in coordination with the tail sequence output. This means that RejSamp cannot begin until the entire head sequence has been output, and that unnecessary memory accesses are incurred. To address this, we exploit the property that AES-CTR can generate output blocks independently. The proposed architecture adjusts the AES-CTR counter to generate the tail sequence in advance and preloads a dense vector of $\mathbb{F}_q$ elements into the buffer. Subsequently, the generated head sequence is fed directly into RejSamp on the fly, without being saved to memory. This enables head sequence generation and RejSamp to be processed in a pipelined fashion, reducing both pseudorandom number generation overhead and unnecessary memory accesses. It should be noted that this approach cannot be applied to the SHAKE256-based XOF that generates output sequentially from the beginning since pre-generation of the tail is not possible.

Symmetric matrix expansion In QR-UOV, the coefficient sequence of the symmetric matrix $\bar{P}_{i,1}$ is generated by the PRG only for the entries corresponding to the upper triangular part. If this output is stored in memory as-is, the number of stored coefficients can be reduced. However, this is inefficient from the perspective of matrix operations since



(a) Expansion for non-diagonal blocks. (b) Expansion for diagonal blocks.

Figure 3: Overview of symmetric matrix expansion ($\widetilde{V} = D\lceil V/D \rceil$).

block matrices cannot be retrieved as contiguous units. It is useful to expand the upper triangular entries into a matrix format for the subsequent matrix operations.

Let $\widetilde{V} := D\lceil V/D \rceil$. The proposed architecture allocates a memory region capable of holding $\widetilde{V} \times \widetilde{V}$ polynomials for $\bar{P}_{i,1}$, where the original $V \times V$ matrix is padded so that both dimensions are multiples of D. The PRG output is written directly to the upper-triangular entries of the symmetric matrix. Here, the write destination address and the number of polynomials to be stored for each entry are stored in a ROM in advance, and the data is written to memory based on the stored address. Subsequently, the lower triangular entries are expanded by exploiting the symmetry of the matrix.

Figure 3 illustrates an overview of the symmetric matrix expansion in units of block matrices. The proposed architecture expands the padded $\widetilde{V} \times \widetilde{V}$ symmetric matrix in units of $D \times D$ block matrices A_{ij} $\left(i, j \in \{1, 2, \ldots, \widetilde{V}/D\}\right)$. For non-diagonal block matrices A_{ij} with $i > j$, A_{ji} stored in the upper triangular region is transposed and assigned using the relation $A_{ij} = A_{ji}^{\top}$. For diagonal block matrices A_{ii}, the lower triangular entries within the block are restored by exploiting the symmetry with respect to the diagonal entries inside the block.

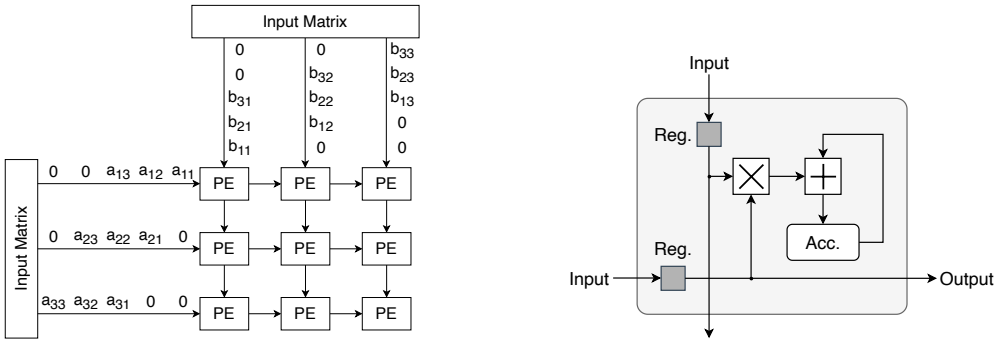
In a straightforward $D \times D$ block-matrix expansion, reading a block matrix requires D clock cycles, and writing the expanded matrix requires an additional D clock cycles. In contrast, the proposed architecture employs two expansion units for $D \times D$ blocks; while one unit reads the next block, the other writes the block expanded in the previous iteration, thereby improving the throughput of the expansion process.

3.3 Polynomial-matrix operations

The polynomial-matrix operations performed in QR-UOV are multiplication, addition/subtraction, and comparison. Among these, addition/subtraction and comparison can be performed independently on elements at the same position. On the other hand, matrix multiplication requires multiply-accumulate operations involving multiple elements spanning the row and column directions of the input matrices in order to compute a single output element. The proposed architecture is equipped with a dedicated arithmetic unit specialized for computationally intensive polynomial-matrix multiplication, as well as a separate unit for element-wise operations.

In addition, for Verify, where each public-key component is evaluated as a quadratic form, the proposed architecture employs a direct-accumulation datapath that avoids explicitly computing and storing intermediate vector–matrix products, thereby enabling further acceleration of Verify.

Polynomial-matrix multiplication The proposed architecture parallelizes polynomial-matrix multiplication using a systolic array structure, which is a widely adopted approach for matrix multiplication. A systolic array [KL78] is an arithmetic unit in which processing



(a) 3×3 systolic array.

(b) Internal structure of PE.

Figure 4: Overview of systolic array and PE.

elements (PEs) are connected in a grid pattern, performing computations while transferring data between neighboring PEs. Figure 4a shows an overview of the systolic array, and Figure 4b shows an individual PE, where we feed the elements of input matrices into PEs, shifting one position in the row and column directions per clock cycle. Each PE performs a multiplication operation on the two input values and accumulates the result. Note that the output circuit for retrieving computation results is omitted in Figure 4b for simplicity. In a systolic array, once input data is read from memory, it can be reused multiple times across PEs, which significantly reduces the number of memory accesses. Furthermore, since each PE independently performs multiply-accumulate operations using local inputs and an internal accumulator, data transfer and computation can be executed simultaneously on every clock cycle. Thus, the systolic array achieves high-throughput execution of computationally intensive matrix multiplication.

The proposed architecture employs a systolic array with PEs arranged in a $D \times D$ configuration, where D is an integer. Since QR-UOV handles polynomial matrices over $\mathbb{F}_{q^\ell}$, the inputs and outputs of each PE are polynomials. Each PE consists of a polynomial multiply-accumulate unit over $\mathbb{F}_{q^\ell}$, described in the following, in order to perform polynomial-matrix multiplication. In addition to matrix-matrix multiplication, QR-UOV also requires vector-matrix multiplication. The vector-matrix multiplication is also executed using the same systolic array without adding dedicated circuits. In this case, not all PEs in the $D \times D$ systolic array can be fully utilized simultaneously; however, multiply-accumulate operations for D output elements can be performed in parallel.

Quadratic form operation In Verify, a quadratic form is evaluated over $\mathbb{F}_{q^\ell}$ to compute each component t'_i $(i = 1, \ldots, m)$ of $\boldsymbol{t}' := \mathcal{P}(\boldsymbol{s})$. In general, for a vector $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$ and a matrix $A = (a_{j,k})_{j,k=1}^n$, the quadratic form can be expanded as

$$\boldsymbol{x}^\top A \boldsymbol{x} = \sum_{j=1}^{n} \sum_{k=1}^{n} x_j a_{j,k} x_k.$$

Therefore, $\boldsymbol{x}^\top A \boldsymbol{x}$ is computed by evaluating the term $x_j a_{j,k} x_k$ for each matrix element $a_{j,k}$ and accumulating the results.

This quadratic form is usually computed by first performing the vector-matrix multiplication $\boldsymbol{x}^\top A$, followed by an inner product operation using the result. However, the naive approach requires the intermediate multiplication result to be temporarily stored and subsequently read back. In contrast, the proposed architecture employs a dedicated

datapath that directly evaluates the quadratic form $x_j a_{j,k} x_k$. In the evaluation, D polynomial elements of the matrix A are read out and combined with the pre-buffered vector $\boldsymbol{x}$ to compute $x_j \cdot a_{j,k+l} \cdot x_{k+l}$ $(l = 0, \ldots, D-1)$ in a D-way parallel manner. The resulting D products are then combined using an adder tree. This allows $\boldsymbol{x}^\top A \boldsymbol{x}$ to be accumulated directly without storing any intermediate results.

Other polynomial operations The proposed hardware selectively performs addition/subtraction and comparison operations in parallel using D arithmetic units that can switch between these operations. Note that in QR-UOV, the above operations are performed element-wise on matrix entries and have significantly lower computational costs than multiplication.

$\mathbb{F}_{q^\ell}$ arithmetic unit Multiplication over $\mathbb{F}_{q^\ell}$ requires coefficient-wise operations, a convolution operation that accumulates products of multiple coefficients, and a reduction modulo the irreducible polynomial. When multiplying two polynomials of degree at most $\ell - 1$, we first produce a polynomial of degree at most $2(\ell - 1)$ by ordinary polynomial multiplication. Then, we reduce it to a polynomial of degree at most $\ell - 1$ modulo the irreducible polynomial. In this process, it is necessary to accumulate multiple multiplication and addition results over $\mathbb{F}_q$. The proposed method collects all partial products for each output degree and then accumulates them in a single step using a Wallace-tree accumulator without incrementally accumulating partial products at the coefficient product stage. This suppresses the increase in critical path length during the intermediate-value accumulations. On the other hand, other $\mathbb{F}_{q^\ell}$ arithmetic operations are performed by the $\mathbb{F}_q$ arithmetic units described below.

$\mathbb{F}_q$ arithmetic unit Since QR-UOV uses a Mersenne prime $q = 2^n - 1$ as the field characteristic, for an arithmetic result z larger than n bits, letting z_l denote the lower n bits and z_h denote the remaining upper bits, the relation $2^n \equiv 1 \bmod 2^n - 1$ yields

$$2^n z_h + z_l \equiv z_h + z_l \bmod 2^n - 1.$$

This means that z can be reduced by adding z_h to z_l. Furthermore, the negation of an input $x \in [0, q)$ is expressed as

$$-x \equiv x_c \bmod 2^n - 1,$$

where x_c is the value obtained by bitwise inverting the n-bit representation of x, satisfying the relation $x + x_c = 2^n - 1$. This shows that the negation of $x \in [0, q)$ can be realized by inverting all bits of x. Consequently, addition and subtraction over $\mathbb{F}_q$ can be implemented using the simple modular reduction described above and bitwise inversion.

We next describe the construction of the $\mathbb{F}_q$ multiplier using the properties above. For an input $x \in [0, q)$, multiplication by 2^i $(i = 0, 1, \ldots, n-1)$ is given by a left rotation $\mathsf{rotate_left}$ as

$$2^i x \equiv \mathsf{rotate_left}(x, i) \bmod 2^n - 1.$$

From this property, partial products can be reduced modulo $2^n - 1$ in advance by applying left rotation during multiplication. This fixes the number of partial products accumulated at each digit and the bit width of the final addition, reducing latency and simplifying the modular reduction after addition. As with the $\mathbb{F}_{q^\ell}$ multiplier, a Wallace tree is used to accumulate the partial products at each digit in the $\mathbb{F}_q$ multiplier. Computations are carried out using the redundant range $[0, 2^n)$, in which both 0 and q represent the zero element of $\mathbb{F}_q$[3]. This representation simplifies the internal datapath by omitting

[3]In polynomial element comparisons, 0 and q are treated as the same value.

canonicalization after each arithmetic operation. The conversion from $x = q$ to 0 at the external interface output is assumed to be handled by the host software, and this conversion is omitted inside the hardware.

3.4 Linear equation solver

The proposed solver accepts only a full-rank coefficient matrix. If L is detected to be rank deficient, Sign is terminated with a failure, and is then re-executed with a different random seed. This design avoids a variable number of salt-resampling and linear-solving iterations inside the hardware core, allowing each invocation of Sign to follow a fixed execution schedule. The rank-deficient probability is approximately $1/q$. The security implications of this implementation approach are discussed in Section 5.2.

The proposed architecture solves the system of linear equations using Gaussian elimination. In QR-UOV, the system of linear equations to be solved is given over $\mathbb{F}_q$. Previous methods targeting Gaussian elimination over binary and binary-extension fields, including a binary solver over GF(2) and a UOV-oriented solver over GF(2^4) and GF(2^8), have been reported in [BCH+23, WSN16]. The proposed architecture is based on the design by Beullens et al. [BCH+23], with its finite-field arithmetic unit replaced by the $\mathbb{F}_q$ arithmetic unit for the Mersenne prime $q = 2^n - 1$ described in Section 3.3. Since QR-UOV does not require matrix operations over $\mathbb{F}_q$ other than solving the linear system, the arithmetic unit adopts a simplified systolic array specific to Gaussian elimination.

3.5 Internal memory structure

The internal memory is used to transfer data among the seed expansion unit, the polynomial-matrix operation unit, and the linear equation solver unit. In the proposed architecture, data that must be retained throughout the entire processing or across multiple loop iterations—such as seeds and secret-key matrices—are stored separately from data that can be overwritten on a per-loop or per-operation basis—such as public-key matrices and intermediate values. In addition, to supply data regularly to the $D \times D$ systolic array, the polynomial-matrix regions are organized so that D polynomials can be read and written as a single unit. Consequently, the number of coefficients stored per unit is ℓD, and since each coefficient over $\mathbb{F}_q$ is generated from one byte of data, the memory width is $8\ell D$ bits.

In QR-UOV, for the public-key components $\bar{P}_i$, the matrices $\bar{P}_{i,1}$ and $\bar{P}_{i,2}$ are expanded by $\mathsf{Expand}_{\mathsf{pk}}(\mathsf{seed}_{\mathsf{pk}}, i)$ with the index $i = 1, \ldots, m$ fixed within each iteration, different from UOV and MAYO, which generate the matrices in an order such that $P_1[a,b], P_2[a,b], \ldots, P_m[a,b]$ are produced consecutively for each matrix element position (a,b). The expanded public-key matrices can therefore be used for the following operations within the same loop iteration. Exploiting this property, we can reuse the memory region storing the expanded public-key matrix in the next loop iteration once the current iteration is complete, which indicates that at most one iteration's amount of public-key data needs to be retained in on-chip memory at any time. In addition, it is not necessary to hold the values of $\bar{P}_{i,3}$ for all i simultaneously in on-chip memory. Specifically, in KeyGen, since $\bar{P}_{i,3}$ computed at each loop iteration can be output externally, the same output buffer can be reused for the computation in the next iteration. Similarly, in Verify, only the $\bar{P}_{i,3}$ required for each loop iteration needs to be read from the public key. In this way, generating or outputting $\bar{P}_{i,1}$, $\bar{P}_{i,2}$, and $\bar{P}_{i,3}$ on a per-iteration basis makes it possible to avoid retaining the entire public-key data in on-chip memory.

Table 2 shows the memory segmentation of the proposed architecture. The internal memory consists of eight MEMs of equal depth, where the depth is determined by MEMs 2 and 3, which store the largest amount of data. MEMs 2 and 3 primarily store the public-key matrix $\bar{P}_{i,1} \in \mathbb{F}_{q^\ell}^{V \times V}$, and the required depth for $\bar{P}_{i,1}$ is expressed as $\lceil V/D \rceil \times \widetilde{V}$. In the proposed architecture, values generated within the processing loop over the public-key

Table 2: Memory segmentation

MEM	Main stored data
0	Seeds, messages, signatures, and other I/O data
1	$\bar{S}'$ and data generated outside the loop
2, 3	Public-key matrix $\bar{P}_{i,1}$ (double buffer)
4, 5	Public-key matrix $\bar{P}_{i,2}$ (double buffer)
6, 7	Intermediate matrices within the loop (double buffer)



(a) Naive sequential execution. (b) Proposed pipelined execution.

Figure 5: Timelines of public-key expansion and polynomial-matrix operations.

matrix P_i are stored using a double buffer, thereby enabling the overlap of public-key expansion and polynomial-matrix operations. This scheduling is described in Section 3.6.

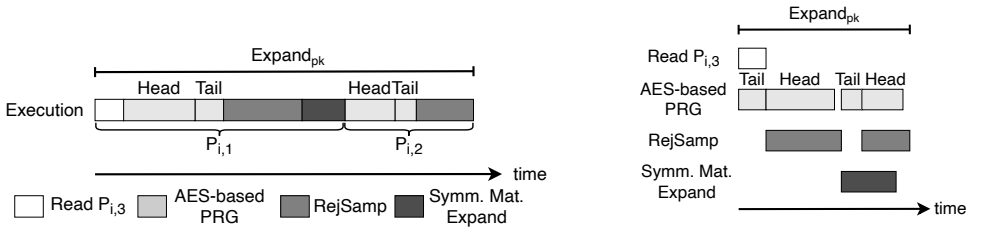
3.6 Scheduling

The proposed hardware executes KeyGen, Sign, and Verify of QR-UOV by coordinating the arithmetic units and internal memory organization according to the following scheduling strategies.

Pipelined execution of public-key expansion and polynomial-matrix operations In QR-UOV, KeyGen, Sign, and Verify each execute a loop of m iterations that processes the public-key components $\bar{P}_i$. The operations in each loop iteration can be divided into public-key expansion via $\mathsf{Expand}_{\mathsf{pk}}(\mathsf{seed}_{\mathsf{pk}}, i)$ and polynomial-matrix operation using the expanded public-key matrices.

Figure 5 illustrates the timelines of public-key expansion and polynomial-matrix operations, where (a) and (b) indicate naive sequential execution and the proposed pipelined execution, respectively. The proposed hardware is equipped with a double buffer for the public-key matrices described in Section 3.5, which enables the $(i+1)$-th public-key matrix elements to be expanded into one buffer simultaneously, while the i-th public-key matrix elements, stored in the other buffer, are supplied to the polynomial-matrix operation unit. Therefore, after the first public-key expansion is complete, the i-th polynomial-matrix operation and the $(i+1)$-th public-key expansion are executed in parallel. As a result, over the $m-1$ iterative operations excluding the first public-key expansion and the last polynomial-matrix operation, the shorter of the two processes is hidden, reducing the total execution time by the equivalent of $m-1$ operations compared to a naive sequential execution as shown in Figure 5(a).

Pipelined execution within public-key expansion Figure 6 shows the timeline inside the public-key expansion during the processing of the public-key component $\bar{P}_i$, where (a) and (b) indicate naive sequential execution and the proposed pipelined execution, respectively. The public-key expansion consists of the generation of $\bar{P}_{i,1}$ and $\bar{P}_{i,2}$ via the PRG and RejSamp, followed by the symmetric matrix expansion of $\bar{P}_{i,1}$. In the proposed hardware, thanks to the on-the-fly AES-CTR-based RejSamp described in Section 3.2, the PRG-based head sequence generation and RejSamp can be executed in parallel. Furthermore, since the symmetric matrix expansion of $\bar{P}_{i,1}$ is independent of the generation of $\bar{P}_{i,2}$, the



(a) Naive execution.

(b) Proposed pipelined execution.

Figure 6: Timelines in public-key expansion.

generation of $\bar{P}_{i,2}$ can begin while the symmetric matrix expansion block is filling in the lower triangular entries after the upper triangular entries of $\bar{P}_{i,1}$ have been generated. In Verify, the public-key expansion also includes the read-out of $\bar{P}_{i,3}$ contained in the public key; however, this read-out process is independent of the expansion of $\bar{P}_{i,1}$ and $\bar{P}_{i,2}$ generated from $\mathsf{seed}_{\mathsf{pk}}$. Therefore, the read-out of $\bar{P}_{i,3}$ used in the same loop iteration can also be executed in parallel with the expansion of $\bar{P}_{i,1}$. In this way, by executing computations in parallel, the processing time for public-key expansion is reduced compared to a naive sequential execution.

4 Performance evaluation

This section evaluates the performance of the proposed hardware through FPGA implementation. We target the recommended parameters of QR-UOV, $q = 127$ and $\ell = 3$, and evaluate each parameter set corresponding to SL-I, SL-III, and SL-V. The systolic array size of the proposed hardware is set to $D = 6$.

The FPGA platform is the Xilinx Artix-7 xc7a200tfbg676-3, which is commonly used as a benchmarking hardware platform for PQC schemes, and Vivado 2025.2 is used for logic synthesis, place-and-route, and simulation. We evaluate the clock cycle count, maximum operating frequency, and circuit area for KeyGen, Sign, and Verify at each security level. The execution time is calculated by dividing the measured clock cycle count by the maximum operating frequency. For Sign, the execution time corresponds to that required for a single attempt assuming the linear system is full rank, in accordance with the evaluation used for other multivariate schemes such as UOV. The maximum operating frequency and circuit area are obtained from the post-place-and-route analysis results. Note that the circuit area includes only the QR-UOV core itself, and excludes the host interface and external memory controller.

Table 3 shows the performance of the proposed hardware, where the execution times are on the order of milliseconds. For comparison, the table also includes the performance results of existing UOV and MAYO hardware implementations. The proposed hardware achieved latencies below 1 ms for all three operations at SL-I. As the security level increases, the QR-UOV parameters V and M grow larger, resulting in an increase in clock cycle count (and hence execution time) for all operations. In particular, KeyGen takes longer than Sign and Verify because the computation of the public-key component $\bar{P}_{i,3}$ involves polynomial matrix-matrix multiplication, and the increase rate in clock cycle count with respect to security level is also larger for KeyGen.

Table 4 shows the resource utilization of the major units at each security level. Here, BRAM utilization increases significantly as the security level rises, whereas the increases in LUT and FF usage are moderate. This is because the sizes of the public-key matrices, intermediate matrices, and data for linear equation solving that must be retained grow

Table 3: Performance and resource utilization comparison with existing hardware

SL	Design	Clock cycles / Time [ms]			$F_{\max}$ [MHz]	Resource utilization			
		KeyGen	Sign	Verify		LUT	FF	BRAM	DSP
I	This Work	162k / 0.74	62k / 0.28	42k / 0.19	217.9	88k	43k	25.5	0
	UOV†	3.75M / 40.93	303k / 3.30	61k / 0.67	91.8	38k	26k	48.0	2
	MAYO‡	5.80k / 0.08	31k / 0.43	4.59k / 0.05	75	106k	38k	45.5	2
III	This Work	793k / 3.74	175k / 0.83	128k / 0.60	211.9	99k	46k	42.0	0
	UOV†	16.46M / 174.95	1.20M / 12.75	196k / 2.08	94.1	43k	32k	184.5	4
	MAYO‡	12k / 0.14	66k / 0.65	9.03k / 0.10	100	169k	60k	96.0	2
V	This Work	2.05M / 9.77	368k / 1.75	278k / 1.32	210.0	101k	46k	75.5	0
	UOV†	38.40M / 414.73	2.65M / 28.57	364k / 3.93	92.6	83k	41k	359.0	4
	MAYO‡	20k / 0.20	111k / 1.12	15k / 0.14	100	246k	83k	194.5	2

Platform: Artix-7, except for MAYO SL-III and SL-V (Kintex-7). †[BCH+23]. ‡[HSK+24].

Table 4: Resource utilization for SL-I, SL-III, and SL-V

Component	SL-I			SL-III			SL-V		
	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF	BRAM
AES-based PRG	21.3k	9.7k	4	27.9k	11.7k	4	32.4k	11.1k	4
⌞ Unrolled AES	9.4k	5.4k	0	13.2k	7.3k	0	12.9k	6.7k	0
⌞ RejSamp	6.1k	3.5k	2	6.1k	3.4k	2	6.0k	3.4k	2
Poly. Mat. Mul.	29.6k	9.2k	0	30.9k	9.3k	0	29.3k	9.2k	0
⌞ Systolic Array	27.4k	6.3k	0	28.6k	6.4k	0	27.1k	6.3k	0
SHAKE-based PRG	9.9k	6.9k	3	10.2k	6.9k	3	10.0k	6.9k	3
⌞ SHAKE	4.4k	3.0k	1	4.6k	3.0k	1	4.5k	3.0k	1
⌞ RejSamp	4.1k	3.2k	2	4.2k	3.2k	2	4.1k	3.2k	2
Symm. Mat. Expand	2.6k	1.9k	0	2.6k	1.9k	0	2.6k	1.9k	0
Linear Eq. Solver	9.0k	5.8k	2.5	9.2k	5.9k	3	9.2k	5.9k	4.5
Quad. Form Ops.	7.9k	4.6k	0	8.8k	5.6k	0	9.2k	6.7k	0
Memory Buffer	5.5k	2.5k	16	5.5k	2.5k	32	5.6k	2.6k	64
Others	2.4k	2.3k	0	3.5k	2.5k	0	2.3k	2.2k	0
Total	88,015	42,994	25.5	98,534	46,490	42	100,504	46,397	75.5

with the QR-UOV parameters V and M, while the arithmetic units are shared across all security levels. Consequently, the increase in circuit area due to higher security levels manifests primarily as an increase in on-chip memory capacity rather than the datapath. DSP usage is zero at all security levels. This is because the arithmetic units for $\mathbb{F}_q$ and $\mathbb{F}_{q^\ell}$ in the proposed hardware are constructed primarily as LUT-based logic circuits.

Table 5 shows the breakdown of clock cycle counts when public-key expansion and polynomial-matrix operations at SL-I are executed independently, in order to examine the execution time of the major operations in QR-UOV. In KeyGen and Sign, polynomial-matrix multiplication is the dominant operation, whereas in Verify, quadratic-form operations account for the largest fraction. In KeyGen, polynomial-matrix operations account for 76.1% of the total, with multiplication alone accounting for 69.6%, indicating that the polynomial-matrix multiplication required to compute $\bar{P}_{i,3}$ is dominant. In Sign and Verify, public-key expansion accounts for 38.0% and 47.6% of the total, respectively. These results demonstrate that, in order to accelerate QR-UOV, it is important not only to speed up polynomial-matrix operations over $\mathbb{F}_{q^\ell}$, but also to reduce the overhead arising from public-key expansion. This finding validates the approach adopted in the proposed hardware, which parallelizes polynomial-matrix multiplication using a $D \times D$ systolic array and introduces a quadratic form operation unit, acceleration of the public-key expansion, and its parallel execution with polynomial-matrix operations.

Table 3 also shows a comparison with existing hardware of other multivariate schemes, namely UOV and MAYO. For UOV, we refer to the hardware corresponding to I-pkc+skc,

Table 5: Breakdown of cycle counts for KeyGen, Sign, and Verify at SL-I

Component	KeyGen	Sign	Verify
Public-key Expansion	37,692 (19.0%)	37,692 (38.0%)	37,705 (47.6%)
Poly. Mat. Ops.	151,416 (76.1%)	56,322 (56.8%)	40,861 (51.6%)
⌞ Multiplication	138,348 (69.6%)	53,784 (54.2%)	–
⌞ Element-wise Ops.	13,068 (6.6%)	2,538 (2.6%)	–
⌞ Quad. Form Ops.	–	–	40,861 (51.6%)
Gaussian Elimination	–	3,484 (3.5%)	–
Others	9,783 (4.9%)	1,726 (1.7%)	586 (0.7%)
Total	198,891	99,224	79,152

The cycle counts in this table are measured values obtained when public-key expansion and polynomial-matrix operations are executed separately.

III-pkc+skc, and V-pkc+skc reported in [BCH+23]. For MAYO, we refer to the hardware of [HSK+24], which corresponds to the Round 2 specification [BCC+24]. Although another MAYO hardware exists in [SMA+24], it is omitted here as its algorithm and parameters are based on the first version [Beu22]. Both the UOV and MAYO implementations employ a pipelined AES implementation for key expansion, as does the proposed hardware.

Compared to UOV, QR-UOV achieves faster execution in all of KeyGen, Sign, and Verify; at SL-I, the speedups are 55.3×, 11.8×, and 3.5×, respectively. On the other hand, QR-UOV consumes more LUTs than UOV, owing to the $D \times D$ systolic array incorporated in the proposed hardware to parallelize polynomial-matrix multiplication. Nevertheless, in terms of the LUT-based area-time product (ATP) computed from Table 3, QR-UOV for SL-I achieved 65.1, 24.6, and 16.7 [$\times 10^3$ LUT $\cdot$ ms] for KeyGen, Sign, and Verify, respectively, outperforming the corresponding UOV values of 1,555.3, 125.4, and 25.5.

Compared to MAYO, the execution time of QR-UOV is longer in most cases. On the other hand, the MAYO hardware prioritizes high throughput and requires more LUTs and BRAMs. QR-UOV achieves a lower ATP than MAYO (45.6 [$\times 10^3$ LUT $\cdot$ ms]) in the Sign operation for SL-I. Similar results were observed for SL-III and SL-V. In summary, QR-UOV exhibits a lower ATP than UOV, and also demonstrates favorable area-time efficiency in Sign compared to MAYO.

5 Discussion

5.1 Other parameter sets

While the proposed hardware in this paper targets the recommended QR-UOV parameters $(q, \ell) = (127, 3)$, QR-UOV also supports the parameter sets $(q, \ell) = (7, 10)$, $(31, 3)$, and $(31, 10)$. In the following, we describe the applicability of the proposed architecture to these parameter sets.

For $q = 7$ and $q = 31$, the number of partial products generated during $\mathbb{F}_q$ multiplication decreases, thereby reducing the number of LUTs required for the $\mathbb{F}_q$ multiplier. If $\mathbb{F}_q$ elements are handled as 1-byte data, the internal memory size remains equivalent to that for $q = 127$. However, since the number of bits discarded when extracting $\mathbb{F}_q$ elements from the PRG output increases compared to the case of $q = 127$, it would be effective to optimize the storage scheme in accordance with the data width required for each $\mathbb{F}_q$.

For $\ell = 10$, multiplication over $\mathbb{F}_{q^\ell}$ involves the multiplication of two degree-9 polynomials, requiring 100 $\mathbb{F}_q$ multiplications. Since the permissible systolic array size D is heavily influenced by the circuit area of the $\mathbb{F}_{q^\ell}$ multiplier, the value of D would likely need to be reduced compared with that for $\ell = 3$. Furthermore, since the data size per polynomial increases, the internal memory data width must be widened to process a large

number of operations at once, which is expected to result in increased BRAM utilization.

5.2 Side-channel resistance

In the following, we discuss the proposed hardware from the perspective of side-channel attacks. The proposed hardware is designed to achieve high-speed execution of QR-UOV, and accordingly, a constant-latency design is adopted in which each internal operation completes within a predetermined number of clock cycles. Specifically, the arithmetic units, memory accesses, and Sign follow fixed schedules, ensuring that the number of execution cycles does not vary depending on the values of the data being processed. Note, however, that this property does not claim resistance against side-channel attacks such as power analysis, electromagnetic analysis, or fault attacks.

Hardware targeting side-channel attack resistance must apply countermeasures such as masking and hiding. In UOV and its derivatives, the secret key expansion process and the resulting secret key coefficients are the primary targets of physical attacks such as side-channel attacks. In particular, for the PRG of QR-UOV used in secret key expansion, masked AES/SHAKE would be necessary. Vinegar variables generated for each signing should also be treated as secret information. In fact, a single-trace side-channel attack that recovers vinegar variables of UOV Sign and subsequently recovers the secret key was reported in [ACK+23]. The linearization matrix L derived from the secret information, as well as the operations performed on it, should also be considered secret-dependent.

QR-UOV has several additional implementation notes in comparison with UOV. First, the secret key expansion of QR-UOV employs RejSamp to convert PRG output into elements of $\mathbb{F}_q$. This process requires the detection of rejected candidate values and the selection and packing of accepted values. In software implementations, branches, memory accesses, or packing operations that depend on the rejection positions can become sources of side-channel leakage, making the conventional strict constant-time implementation non-trivial. In contrast, the proposed hardware assumes fixed-latency BRAM accesses and achieves RejSamp with a fixed schedule.

Second, when considering masking countermeasures, an important question is whether the rejection pattern must be treated as secret information. Since the rejection positions are statistically independent of the accepted $\mathbb{F}_q$ coefficients, observing the rejection pattern does not directly reveal the accepted values. If the rejection pattern can thus be treated as public control information, the packing of accepted values can be implemented as public control flow, potentially reducing the cost of side-channel-resistant rejection sampling.

Third, the QR-UOV Sign fixes the vinegar variables and resamples the salt, repeating until the linear equation $Lx = \boldsymbol{t} - \boldsymbol{u}$ becomes solvable, which ensures EUF-CMA security. Here, L is a secret-dependent matrix derived from the secret key and the per-signature vinegar variables. Let $d = m - \mathrm{rank}(L)$. The probability that the equation has a solution for a randomly determined right-hand side is q^{-d}. Consequently, the number of iterations depends on the rank deficiency of L, which may leak information derived from the secret key or the per-signature vinegar variables.

To avoid leakage of secret-dependent information through the number of salt retries, the proposed hardware adopts a variant in which Sign terminates with a failure if L is not full rank. This approach corresponds to that of UOV/MAYO, where rank-deficient instances are discarded and signing is restarted with newly sampled randomness. Under this approach, since no signature is output for non-full-rank attempts and the used vinegar variables are discarded, secret information leakage related to the vinegar variables is avoided. Furthermore, by restricting the linear solver to full-rank systems only, the fixed-schedule Gaussian elimination can be used. Signatures generated by this variant can be verified using the standard QR-UOV verification algorithm.

6 Conclusion

In this paper, we proposed a hardware accelerator that efficiently executes key generation, signature generation, and signature verification of QR-UOV at each security level. QR-UOV requires polynomial-matrix operations over $\mathbb{F}_{q^\ell}$ based on the quotient polynomial ring structure, coefficient generation for the Mersenne prime field $\mathbb{F}_q$, and the expansion of symmetric matrices generated in compressed form, all of which demand a datapath different from those of existing hardware implementations for UOV and MAYO. To address these requirements, the proposed hardware introduces polynomial-matrix multiplication via a $D \times D$ systolic array, a direct accumulation of quadratic forms for signature verification, on-the-fly rejection sampling that exploits the block independence of AES-CTR, block-wise symmetric matrix expansion, and pipelined execution of public-key expansion and polynomial-matrix operations. Through FPGA evaluation, we confirmed that, for the recommended parameters $(q, \ell) = (127, 3)$ at security level I, key generation, signature generation, and signature verification were executed in 0.74 ms, 0.28 ms, and 0.19 ms, respectively. Furthermore, from a comparison with existing UOV hardware implementations, we confirmed that the proposed architecture achieved a superior LUT-based area-time product, demonstrating favorable area-time efficiency as a hardware accelerator for multivariate schemes. Future work includes the application of the proposed architecture to other QR-UOV parameters, a detailed evaluation of the area-performance trade-off including the systolic array size, and the introduction of countermeasures against physical attacks including side-channel attacks.

A Effects of proposed acceleration techniques

Table 6 shows the incremental effects of the proposed acceleration techniques at SL-I. M1–M4 are applied cumulatively, and the final configuration, denoted by All, additionally incorporates pipelined execution of public-key expansion and polynomial-matrix operations.

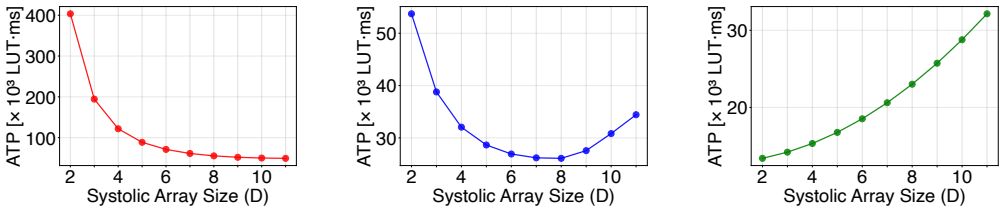
Table 6: Effects of proposed acceleration techniques for SL-I

Technique	Clock cycles / Time [ms]			$F_{\max}$ [MHz]	Resource utilization		
	KeyGen	Sign	Verify		LUT	FF	BRAM
M1	247,468 / 1.25	147,781 / 0.75	149,043 / 0.75	197.7	67,735	34,868	21.5
M1+M2	214,389 / 1.11	114,722 / 0.59	116,103 / 0.60	193.0	74,143	38,359	24.5
M1+M2+M3	214,389 / 0.99	114,722 / 0.53	103,992 / 0.48	217.5	88,713	43,025	26.5
M1+M2+M3+M4	198,891 / 0.90	99,224 / 0.45	79,152 / 0.36	221.2	88,733	42,963	26.5
All	161,729 / 0.74	61,990 / 0.28	42,142 / 0.19	217.9	88,015	42,994	25.5

M1: High-speed Rejection Sampling. M2: on-the-fly AES-CTR-based Rejection Sampling.
M3: Direct quadratic-form accumulation for Verify. M4: Pipelined execution within public-key expansion.
All: M1–M4 + Pipelined execution of public-key expansion and polynomial-matrix operations.

B Effects of systolic-array size

Figure 7 estimates the LUT-based ATP at SL-I for different values of D. We assume that the cycle counts of matrix–matrix and matrix–vector multiplication scale as $1/D^2$ and $1/D$, respectively, that the LUT usage of the systolic array scales as D^2, and that the maximum operating frequency is 200 MHz. Increasing D improves the ATP of KeyGen, while Sign reaches its minimum near $D = 6$. In contrast, the ATP of Verify increases with D because Verify uses the quadratic-form operation unit rather than the systolic array.



(a) KeyGen. (b) Sign. (c) Verify.

Figure 7: Estimated LUT-based ATP for different systolic-array sizes D.

References

[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 TCHES*, 2023(3):221–245, 2023.

[AFI+24] Rika Akiyama, Hiroki Furue, Yasuhiko Ikematsu, Fumitaka Hoshino, Koha Kinjo, Haruhisa Kosuge, Satoshi Nakamura, Shingo Orihara, Tsuyoshi Takagi, and Kimihiro Yamakoshi. QR-UOV. Technical report, National Institute of Standards and Technology, 2024. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures.

[AUH26] Hiroshi Amagasa, Rei Ueno, and Naofumi Homma. AVX2 implementation of QR-UOV for modern x86 processors. *IACR TCHES*, 2026(2):325–346, 2026.

[BCC+24] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. MAYO. Technical report, National Institute of Standards and Technology, 2024. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures.

[BCD+24] 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. Technical report, National Institute of Standards and Technology, 2024. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures.

[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 TCHES*, 2023(3):321–365, 2023.

[Beu22] Ward Beullens. MAYO: Practical post-quantum signatures from oil-and-vinegar maps. In Riham AlTawy and Andreas Hülsing, editors, *SAC 2021*, volume 13203 of *LNCS*, pages 355–376. Springer, Cham, September / October 2022.

[FG18] Ahmed Ferozpuri and Kris Gaj. High-speed FPGA implementation of the NIST Round 1 Rainbow signature scheme. In David Andrews, René Cumplido, Claudia Feregrino, and Dirk Stroobandt, editors, *2018 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2018, Cancun, Mexico, December 3-5, 2018*, pages 1–8. IEEE, 2018.

[FIKT21] Hiroki Furue, Yasuhiko Ikematsu, Yutaro Kiyomura, and Tsuyoshi Takagi. A new variant of unbalanced Oil and Vinegar using quotient ring: QR-UOV. In

Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 187–217. Springer, Cham, December 2021.

[HL06] Yedidya Hilewitz and Ruby B. Lee. Fast bit compression and expansion with parallel extract and parallel deposit instructions. In *2006 IEEE International Conference on Application-Specific Systems, Architecture and Processors (ASAP 2006), 11-13 September 2006, Steamboat Springs, Colorado, USA*, pages 65–72. IEEE Computer Society, 2006.

[Hsi12] Homer Hsing. Tiny AES. https://opencores.org/projects/tiny_aes, 2012. OpenCores project, accessed 2026-07-09.

[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 Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 3421–3435. ACM Press, October 2024.

[IKA+25] Malik Imran, Safiullah Khan, Zain Ul Abideen, Ciara Rafferty, Ayesha Khalid, Muhammad Rashid, and Máire O'Neill. RejSCore: Rejection sampling core for multivariate-based public key cryptography. In *Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2025, Nanjing, China, December 19-21, 2025*, pages 1–6. IEEE, 2025.

[KL78] H. T. Kung and Charles E. Leiserson. Systolic Arrays for (VLSI). Technical Report CMU-CS-79-103, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, PA, 1978.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar signature schemes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 206–222. Springer, Berlin, Heidelberg, May 1999.

[MAB+26] Dustin Moody, Gorjan Alagic, Maxime Bros, Pierre Ciadoux, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Rene Peralta, Ray Perlner, Angela Robinson, Hamilton Silberg, Daniel Smith-Tone, and Noah Waller. Status report on the second round of the additional digital signature schemes for the NIST post-quantum cryptography standardization process, May. 2026.

[RB20] Sujoy Sinha Roy and Andrea Basso. High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware. *IACR TCHES*, 2020(4):443–466, 2020.

[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 Romain Wacquez and Naofumi Homma, editors, *COSADE 2024*, volume 14595 of *LNCS*, pages 240–259. Springer, Cham, April 2024.

[WSN16] Wen Wang, Jakub Szefer, and Ruben Niederhagen. Solving large systems of linear equations over GF(2) on fpgas. In Peter M. Athanas, René Cumplido, Claudia Feregrino, and Ron Sass, editors, *International Conference on ReConFigurable Computing and FPGAs, ReConFig 2016, Cancun, Mexico, November 30 - Dec. 2, 2016*, pages 1–7. IEEE, 2016.