2023 24th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-3475-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISQED57927.2023.10129320

Novel Implementation of High-Performance Polynomial Multiplication for Unified KEM Saber based on TMVP Design Strategy

Pengzhou He and Jiafeng Xie* (*: corresponding author)
Department of Electrical and Computer Engineering, Villanova University, Villanova PA 19087, USA
Email: {phe,jiafeng.xie}@villanova.edu

***Abstract*—The rapid advancement in quantum technology has initiated a new round of exploration of efficient implementation of post-quantum cryptography (PQC) on hardware platforms. Key encapsulation mechanism (KEM) Saber, a module lattice-based PQC, is one of the four encryption scheme finalists in the third-round National Institute of Standards and Technology (NIST) standardization process. In this paper, we propose a novel Toeplitz Matrix-Vector Product (TMVP)-based design strategy to efficiently implement polynomial multiplication (essential arithmetic operation) for KEM Saber. The proposed work consists of three layers of interdependent efforts: (i) first of all, we have formulated the polynomial multiplication of KEM Saber into a desired mathematical form for further developing into the proposed TMVP-based algorithm for high-performance operation; (ii) then, we have followed the proposed TMVP-based algorithm to innovatively transfer the derived algorithm into a unified polynomial multiplication structure (fits all security ranks) with the help of a series of algorithm-to-architecture co-implementation/mapping techniques; (iii) finally, detailed implementation results and complexity analysis have confirmed the efficiency of the proposed TMVP design strategy. Specifically, the field-programmable gate array (FPGA) implementation results show that the proposed design has at least less 30.92% area-delay product (ADP) than the competing ones.**

***Index Terms*—High-performance, key encapsulation mechanism (KEM) Saber, polynomial multiplication, post-quantum cryptography (PQC), Toeplitz Matrix-Vector Product (TMVP).**

I. Introduction

It has been proven that the well-established quantum computer employing Shor's algorithm can break the current public-key cryptosystems such as Rivest Shamir Adleman (RSA) and Elliptic Curve Cryptography [1], [2], [3]. Indeed, along with the rapid advancement in quantum computing, the attention from the research community on post-quantum cryptography (PQC) and related implementations have reached an all-time high [2]. As indicated by the National Institute of Science and Technology (NIST) third-round PQC standardization process [3], lattice-based cryptography is recognized as one of the most promising classes due to its small implementation complexity and strong security proof [3].

A lot of lattice-based cryptography is based on the learning-with-errors (LWE) problem [4], [5]. The standard LWE-based scheme involves large computational complexity, and hence it takes large resource occupation when implemented on the hardware platform. The learning-with-rounding (LWR) is a variant of LWE [6], and quite a good number of works have been released on the related PQC [7], [8], [9], [10], [11], [12], [13], including the NIST's third-round finalist, the key encapsulation mechanism (KEM) Saber [3].

Existing Works. Overall, we can categorize the existing hardware designs for KEM Saber into two types, namely the hardware-software co-design and the full-hardware design. The former type refers to the recent report of [9], where the authors use the Toom-Cook method to implement the polynomial multiplication for KEM Saber. Another report is recently released in [10], where the authors similarly use the same Toom-Cook approach to obtain better performance at the cost of large resource usage. For the latter type, the first design is released in [11], where the authors use a schoolbook-based polynomial multiplication to obtain an efficient KEM Saber implementation. After that, a Karatsuba algorithm-based KEM Saber is released in [12] to obtain high performance. Optimized polynomial multiplication structures for KEM Saber are recently presented in [14]. Very recently, a compact KEM Saber coprocessor [15] and high-performance KEM Saber architectures are proposed in [16]. These works represent the primary efforts in the field.

Noticing that the polynomial multiplication over ring $\mathbb{Z}_l/(x^N+1)$ (l is either q or p [7], [8]) is the critical arithmetic operation of KEM Saber, the existing works, however, still need significant improvements: (i) the very recent existing high-performance work, such as the structure in [11], is still based on the traditional schoolbook method and hence novel design strategy needs to be developed; (ii) not many algorithm-architecture co-implementation (i.e., mapping algorithm to architecture) techniques have been proposed to improve the performance of the polynomial multiplications, especially on the presenting thorough mathematical derivation process to obtain efficient algorithms for architecture mapping; (iii) there exist limited unified polynomial multiplication works for all three security levels of KEM Saber. With this point of view, in this paper, we propose an efficient implementation of polynomial multiplication for KEM Saber on the field-programmable gate array (FPGA) platform for high-performance applications. Specifically, we have proposed a novel Toeplitz Matrix-Vector Product (TMVP)-based design strategy for the targeted polynomial multiplication that the involved computational complexity is reduced. Meanwhile, the

finalized polynomial multiplication is a unified architecture, which fits well for all three security levels of KEM Saber.

Major Contributions. In total, we have carried out four layers of innovative works for the efficient implementation of polynomial multiplication for KEM Saber, as:

- We have proposed thorough mathematical derivation to lay a solid foundation for the novel TMVP-based design strategy and the related algorithm.
- We have then presented a detailed design process to innovatively transfer the TMVP-originated algorithm into the desired architecture with the help of several algorithm-to-architecture mapping techniques.
- We have conducted complexity analysis and comparison, based on the FPGA implementation results, to demonstrate the efficiency of the proposed TMVP strategy.

Specifically, this is the first hardware implementation of TMVP-based polynomial multiplication for KEM Saber, which offers many unique features: (i) unified structure fits for all three security levels of KEM Saber; (ii) high-performance operation with low computational time; and (iii) overall efficiency in area-time complexities.

The rest of this paper is organized as follows. The preliminaries are introduced in Section II. The proposed TMVP strategy is formulated in Section III. The proposed hardware polynomial multiplication structure is provided in Section IV. Complexity analysis and comparison are presented in Section V. Finally, the conclusions are given in Section VI.

II. Preliminaries

Notations. We follow the existing papers [7], [8], [11] to list the following notations. Define p and q as two powers of 2 as $p = 2^{\varepsilon_p}$ and $q = 2^{\varepsilon_q}$ and let $\mathbb{Z}_q$ be the ring of integers modulo q. We also define the ring of polynomials $\mathcal{R}_p = \mathbb{Z}_p[x]/\langle x^N + 1\rangle$ for p and $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^N+1\rangle$ for q, respectively. **a** is used to represent a vector and $a(x)$ is used to denote the polynomial in $\mathcal{R}$, where the coefficients can be seen as a vector and the i-th coefficient is the i-th entry of the vector. Moreover, the operator $\lfloor\cdot\rceil$ denotes the rounding operation.

Besides that, β_μ represents the binomial distribution based on parameter μ, which produces values in the range of $[-\mu/2, \mu/2]$ with probability of $\frac{\mu!}{(\mu/2+x)!(\mu/2-x)}2^{-\mu}$. $x \leftarrow \beta_\mu$ denotes that x is randomly sampled from the binomial distribution. If we replace x with X, then it means a polynomial X is sampled from the binomial distribution. The notation of $x \leftarrow \mathcal{U}(S)$ denotes that x is uniformly selected from S.

The MLWR-based Encryption Scheme: KEM Saber. The LWR is a variant of the LWE problem, where the error term is introduced by a rounding operation rather than obtaining it from a random distribution [8]. The samples for LWR-based scheme are generated by $(\mathbf{a}, b = \lfloor \frac{p}{q}\langle \mathbf{a}, s\rangle \rceil_p) \in \mathbb{Z}_q^n \times \mathbb{Z}_p$. Saber is built on the hardness of the module LWR (MLWR) problem.

In brief, Saber consists of key generation, encryption, and decryption. In the key generation (Algorithm 1), a public matrix of polynomials **A** and a secret vector of polynomials **s** are generated. Meanwhile, the vector **b** is calculated through the scaling and rounding of the product **As**, where the public key consists of **A** and **b** and the secret key is the vector **s**. In the encryption, the message is encrypted by $v_1{}' = \mathbf{s}'\mathbf{b}^T$ ($\mathbf{s}'$ is a vector for the encryption). The produced ciphertext involves $\mathbf{b}'$ (from rounding $\mathbf{As}'$). While during the decryption, the message is recovered through the approximation of v_1 (from $\mathbf{sb}'$).

Let $\mathcal{F} : \{0,1\}^* \rightarrow \{0,1\}^n$ and $\mathcal{G} : \{0,1\}^* \rightarrow \{0,1\}^{l\times n}$ denote the hash functions SHA3-256 and SHA3-512, respectively, we have Algorithms 1, 2, and 3 to represent KEM Saber.

Algorithm 1 Saber.KEM.PKE.KeyGen() [7], [8]

$(seed_{\mathbf{A}}, \mathbf{b}, \mathbf{s})$=Saber.PKE.KeyGen().
$pk = (seed_{\mathbf{A}}, \mathbf{b})$.
$pkh = \mathcal{F}(pk)$.
$z_1 = \mathcal{U}(\{0,1\}^{256})$.
return $(pk := (seed_{\mathbf{A}}, \mathbf{b}),\ sk := (\mathbf{s}, z_1, pkh))$.

Algorithm 2 Saber.KEM.Encaps($pk = (seed_{\mathbf{A}}, \mathbf{b})$) [7], [8]

$m = \mathcal{U}(\{0,1\}^{256})$.
$(\hat{K}, c) = \mathcal{G}(\mathcal{F}(pk), m)$.
c=Saber.PKE.Enc($pk, m; r$).
$K = \mathcal{F}(\hat{K}, c)$.
return (c, K).

Algorithm 3 Saber.KEM.Decaps($sk = (\mathbf{s}, z, pkh), pk = (seed_{\mathbf{A}}, \mathbf{b}), c$) [7], [8]

m'=Saber.PKE.Dec($\mathbf{s}, c$).
$(\hat{K}', c') = \mathcal{G}(pkh, m')$.
c'=Saber.PKE.Enc($pk, m'; r'$).
if $c = c'$ then return $K = \mathcal{H}(\hat{K}, c)$.
else return $K = \mathcal{H}(z_1, c)$.

Module Ranks. There are three module ranks with $l = 2,\ 3,\ \text{and}\ 4$ for the NIST security levels 1, 3, and 5, respectively, called LightSaber, Saber, and FireSaber. The polynomial degree is set as $N = 256$ and moduli $q = 2^{13}$ and $p = 2^{10}$. The secrets sampled are [-5,5] (LightSaber), [-4,4] (Saber), to [-3,3] (FireSaber), respectively [7], [8].

TMVP over $GF(2^m)$. The TMVP method is originally introduced in [17] for polynomial multiplication over $GF(2^m)$. This approach has been further developed/deployed in following works such as [18], [19], [20], [21]. An $n \times n$ Teoplitz matrix is a matrix $T = [t_{i,j}]_{0\le i,j\le n-1}$, where $t_{i,j} = t_{i-1,j-1}$ [17]. Define $V = (V_0, V_1)$ as an $n \times 1$ column vector (V_0 and V_1 are $\frac{n}{2} \times 1$ column vectors) and T_0, T_1, and T_2 as $\frac{n}{2} \times \frac{n}{2}$ Toeplitz matrices. Thus,

$$\begin{aligned} Z = \begin{bmatrix} Z_0 \\ Z_1 \end{bmatrix} &= \begin{bmatrix} T_0 & T_2 \\ T_1 & T_0 \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \end{bmatrix} \\ &= \begin{bmatrix} T_0(V_0 + V_1) + (T_2 + T_0)V_1 \\ T_0(V_0 + V_1) + (T_1 + T_0)V_0 \end{bmatrix}, \end{aligned} \tag{1}$$

which involves four components, i.e., component matrix point

(CMP), component vector point (CVP), point-wise multiply (PWM), and reconstruction (R): $\mathrm{CMP}(T) = (T_2 + T_0, T_0, T_0 + T_1)$, $\mathrm{CVP}(V) = (V_1, V_0 + V_1, V_0)$, $P = \mathrm{PWM}(\mathrm{CMP}(T), \mathrm{CVP}(V)) = (P_0, P_1, P_2)$, and $Z = \mathrm{R}(P) = (P_0 + P_1, P_1 + P_2)$ (for $P_0 = (T_2 + T_0)V_1$, $P_1 = T_0(V_0 + V_1)$, and $P_2 = (T_1 + T_0)V_0$). Because of this setup, the original complexity of $\mathcal{O}(n^2)$ is now reduced to three matrix-vector products of $\mathcal{O}((\frac{n}{2})^2)$.

The operation of (1) can be iteratively applied to the original polynomial multiplication to obtain subquadratic-complexity. The software implementation of KEM Saber based on TMVP is presented in [22]. However, its strategy (full parallel processing) cannot be directly deployed for hardware implementation (which involves very large resource usage and is impractical in actual applications).

Polynomial Multiplication for KEM Saber. The polynomial multiplication is the key operation of the Saber protocol, where one polynomial involves small-size coefficients (e.g., in the range of -4 to +4 for Saber) and the other polynomial has coefficients of 10-bit or 13-bit (13-bit design covers 10-bit).

III. Algorithmic Formulation

This section focuses on the mathematical formulation for the polynomial multiplication of KEM Saber, i.e., transfer it into the desired form for further extension to derive a novel algorithm originated from the proposed TMVP design strategy.

Extension of the Existing TMVP Approach to the Integers. Define again $t'_{i,j}$ is an integer as well as v'_i. Still,

$$\begin{bmatrix} Z'_0 \\ Z'_1 \end{bmatrix} = \begin{bmatrix} T'_0(V'_0 + V'_1) + (T'_2 - T'_0)V'_1 \\ T'_0(V'_0 + V'_1) + (T'_1 - T'_0)V'_0 \end{bmatrix}, \tag{2}$$

where $t'_{i,j}$ is in the two's complement form. Similarly, the TMVP approach under integers also consists of four components, where two subtractions are used to replace the original additions in the CMP component.

Formulation of the Polynomial Multiplication for KEM Saber. The general form can be defined as follows.

Notation 1. Define polynomials as: $W = \sum_{i=0}^{N-1} w_i x^i$, $D = \sum_{i=0}^{N-1} d_i x^i$, and $G = \sum_{i=0}^{N-1} g_i x^i$, where g_i is the 4-bit coefficient over ring and d_i and w_i are coefficients of 13-bit over ring. Meanwhile, let polynomial W be the product of the polynomials of D and G. We then have ($f(x) = x^N + 1$)

$$W = DG \bmod f(x). \tag{3}$$

Then, we can rewrite (3) into another form of

$$W = G \sum_{i=0}^{N-1} d_i x^i \bmod f(x) = \sum_{i=0}^{N-1} d_i (Gx^i \bmod f(x)). \tag{4}$$

Notation 2. Define again $G^{[0]} = Gx^0 \bmod f(x) = G$, $G^{[1]} = Gx^1 \bmod f(x) = G^{[0]}x \bmod f(x)$, ..., and $G^{[N-1]} = Gx^{N-1} \bmod f(x) = G^{[N-2]}x \bmod f(x)$.

Thus, we can have

$$\begin{aligned} G^{[0]} &= g_0 + g_1 x + g_2 x^2 + \cdots + g_{N-1} x^{N-1}, \\ G^{[1]} &= -g_{N-1} + g_0 x + \cdots + g_{N-2} x^{N-1}, \\ &\cdots \ \cdots \ \cdots \\ G^{[N-1]} &= -g_1 - g_2 x - g_3 x^2 - \cdots + g_0 x^{N-1}, \end{aligned} \tag{5}$$

for $x^N \equiv -1$.

Then, the original multiplication of (4) becomes

$$W = \sum_{i=0}^{N-1} d_i G^{[i]}, \tag{6}$$

which can be further expanded into

$$\begin{aligned} W = \ & w_0 + w_1 x + \cdots + w_{N-1} x^{N-1} \\ = (\ & g_0 d_0 + g_1 d_0 x + \cdots + g_{N-1} d_0 x^{N-1}) \\ + \cdots \ & \cdots \ \cdots \\ + (\ & -g_1 d_{N-1} - g_2 d_{N-1} x - \cdots + g_0 d_{N-1} x^{N-1}), \end{aligned} \tag{7}$$

from where we can observe that each coefficient of W is the addition of N terms (the same order of x), as

$$\begin{aligned} w_0 &= \underbrace{g_0 d_0}_{\text{term}-1} + \underbrace{(-g_{N-1} d_1)}_{\text{term}-2} + \underbrace{(-g_{N-2} d_2)}_{\text{term}-3} + \cdots + \underbrace{(-g_1 d_{N-1})}_{\text{term}-N}, \\ w_1 &= g_1 d_0 + g_0 d_1 + (-g_{N-1} d_2) + \cdots + (-g_2 d_{N-1}), \\ &\cdots \ \cdots \ \cdots \\ w_{N-1} &= g_{N-1} d_0 + g_{N-2} d_1 + g_{N-3} d_2 + \cdots + g_0 d_{N-1}, \end{aligned} \tag{8}$$

where we can further transfer into matrix-vector product as

$$\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \end{bmatrix} = \begin{bmatrix} g_0 & -g_{N-1} & \cdots & -g_1 \\ g_1 & g_0 & \cdots & -g_2 \\ \vdots & \vdots & \ddots & \vdots \\ g_{N-1} & g_{N-2} & \cdots & g_0 \end{bmatrix} \times \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{N-1} \end{bmatrix}, \tag{9}$$

which can be further defined as

$$[W] = [G] \times [D], \tag{10}$$

where $[W]$, $[G]$, and $[D]$ are $N \times 1$, $N \times N$, and $N \times 1$ matrices, respectively, according to (9). Note that the elements in one certain matrix are denoted as $[W]_{i,1}$, $[G]_{i,j}$, and $[D]_{i,1}$ ($1 \le i, j \le N$) such that we can have $[G]_{1,1} = g_0$, $[G]_{2,1} = g_1$, etc.

One can note that $[G]$ is actually an $N \times N$ Toeplitz matrix, one can thus follow the extended TMVP method of (2) to have

$$\begin{bmatrix} W_0 \\ W_1 \end{bmatrix} = \begin{bmatrix} G_0 & G_2 \\ G_1 & G_0 \end{bmatrix} \times \begin{bmatrix} D_0 \\ D_1 \end{bmatrix}, \tag{11}$$

where $[W_i]$ and $[D_i]$ are $\frac{N}{2} \times 1$ matrix-vectors ($0 \le i \le 1$) as $[W_0] = [w_0 \ w_1 \ \cdots \ w_{N/2-1}]^T$, $\cdots$, $[D_1] = [d_{N/2} \ \cdots \ d_{N-1}]^T$; $[G_j]$ are $\frac{N}{2} \times \frac{N}{2}$ Toeplitz matrices ($0 \le j \le 2$) as

$$[G_0] = \begin{bmatrix} g_0 & -g_{N-1} & \cdots & -g_{N/2+1} \\ g_1 & g_0 & \cdots & -g_{N/2+2} \\ \vdots & \vdots & \ddots & \vdots \\ g_{N/2-1} & g_{N/2-2} & \cdots & g_0 \end{bmatrix}, \tag{12}$$

$\cdots \ \cdots \ \cdots$.

Meanwhile, one can also find that $[G_2] = -[G_1]$, such that

$$\begin{aligned} \begin{bmatrix} W_0 \\ W_1 \end{bmatrix} &= \begin{bmatrix} G_0 & -G_1 \\ G_1 & G_0 \end{bmatrix} \times \begin{bmatrix} D_0 \\ D_1 \end{bmatrix} \\ &= \begin{bmatrix} G_0(D_0 + D_1) + (-G_1 - G_0)D_1 \\ G_0(D_0 + D_1) + (G_1 - G_0)D_0 \end{bmatrix}, \end{aligned} \tag{13}$$

TABLE I: Theoretical Computational Complexities of the Schoolbook- and TMVP-based Approaches

Strategy	#Multipliers	#Adders
Schoolbook approach	N^2	N^2
TMVP approach (1-iteration)	$3N^2/4$	$N^2/2+3N/2$

where the original $N \times N$ matrix-vector product (complexity of $\mathcal{O}(N^2)$) has been turned into a form of three $\frac{N}{2} \times \frac{N}{2}$ matrix-vector products (each with a complexity of $\mathcal{O}((\frac{N}{2})^2)$) and extra number of adders, as listed in Table I.

Proposed Mathematical Derivation Strategy. The theoretical analysis (Table I) has shown that the TMVP-based polynomial multiplication has lower complexity than the schoolbook-based one. The direct hardware implementation of (13), however, seems to be impractical, especially considering the fact that the elements of $[G]$ and $[D]$ are 4-bit and 13-bit (or 10-bit) integers over the ring, respectively (the overall resource usage would be huge). Even though we apply the TMVP approach to (13) recursively to obtain subquadratic-complexity, the overall implementation is still quite challenging. It will also potentially have a significant computational delay when taking these recursive operations into account.

With these above considerations, *we thus propose a novel mathematical derivation strategy* to obtain a high-performance realization of the polynomial multiplication with relatively low-complexity: (i) firstly transferring the three $\frac{N}{2}$-size matrix-vector products into the accumulation forms suitable for high-speed operation yet saves computational complexity; (ii) then finding out proper operational sequence among these three accumulation forms to obtain the optimal resource usage; (iii) connecting these related operations to produce the desired final output. Following this strategy, we can thus have:

Step-I. Let us consider $[G_0][D_0+D_1]$ first, which can be easily extended to the other two matrix-vector products. It is obvious that $[D_0+D_1]$ is still an $\frac{N}{2} \times 1$ matrix-vector, and thus we can have ($[G_0][D_0+D_1]$ is an $\frac{N}{2} \times 1$ matrix)

$$\begin{aligned}&[G_0][D_0+D_1] = [G_0]_{:,1}[D_0+D_1]_1 + \\ &[G_0]_{:,2}[D_0+D_1]_2 + \cdots + [G_0]_{:,N/2}[D_0+D_1]_{N/2} \\ &= \sum_{j=1}^{N/2} [G_0]_{:,j}[D_0+D_1]_j,\end{aligned} \tag{14}$$

where the results of the original matrix-vector product (all $N/2$ elements) have become an accumulation of $N/2$ terms of $\sum_{j=1}^{N/2}[G_0]_{i,j}[D_0+D_1]_j$, and each term involves $N/2$ individual point-to-point multiplications of one column $[G_0]_{:,j}$ with the corresponding $[D_0+D_1]_j$.

Note that we propose this kind of accumulation based on multiple considerations (mainly for ease of further implementation). (a) All $N/2$ elements of $[G_0]_{:,j}$ within one accumulation, e.g., $j=1$, actually are the $N/2$ elements of the first column of the matrix $[G_0]$ (from left), which is very easy for mapping into hardware components. (b) The element of $[D_0+D_1]_j$ is one by one processed such that the involved number of adders (13-bit) is minimized, as demonstrated by the hardware design in Section IV.

Similarly, the other two $N/2 \times N/2$ matrix-vector products of (13) can also have

$$\begin{aligned}[-G_1-G_0][D_1] &= \sum_{j=1}^{N/2}[-G_1-G_0]_{:,j}[D_1]_j, \\ [G_1-G_0][D_0] &= \sum_{j=1}^{N/2}[G_1-G_0]_{:,j}[D_0]_j.\end{aligned} \tag{15}$$

Thus, all three $N/2$-size matrix-vector products have been transferred into the forms of accumulation that each requires $N/2$ cycles of operations to obtain the desired output.

Step-II. When connecting (14) and (15) with (13), one can find that the output of (14) needs to be added with the two matrix-vector products of (15) to deliver the final output. Theoretically speaking, there is nothing particular involved within this process. While standing on the implementation point of view, especially that a typical accumulator uses more resources than an adder, we rewrite (13) as

$$\begin{aligned}[W_0] &= [G_0(D_0+D_1) + (-G_1-G_0)D_1], \\ [W_1] &= [G_0(D_0+D_1) + (G_1-G_0)D_0],\end{aligned} \tag{16}$$

and then

$$\begin{aligned}&[G_0(D_0+D_1) + (-G_1-G_0)D_1] \\ &= \sum_{j=1}^{N/2}[G_0]_{:,j}[D_0+D_1]_j + \sum_{j=1}^{N/2}[-G_1-G_0]_{:,j}[D_1]_j \\ &= \sum_{j=1}^{N/2}([G_0]_{:,j}[D_0+D_1]_j + [-G_1-G_0]_{:,j}[D_1]_j),\end{aligned} \tag{17}$$

where the original two separate accumulations are combined together as one to save the resource usage, as shown later also in the implementation section (Section IV).

Similarly, we have

$$\begin{aligned}&[W_1] = [G_0(D_0+D_1) + (G_1-G_0)D_0] \\ &= \sum_{j=1}^{N/2}([G_0]_{:,j}[D_0+D_1]_j + [G_1-G_0]_{:,j}[D_0]_j).\end{aligned} \tag{18}$$

Step-III. It is possible to combine the operations of (17) and (18) together, after considering again that: (i) the obtaining of final output results actually rests on the two accumulations of (17) and (18), which can be processed in parallel; (ii) the actual elements/coefficients involved within (17) and (18) are basically the half matrix of $[G]$ (left side, since only $[G_0]$ and $[G_1]$ are needed) except with some signs inverted, and the elements from the matrix of $[D]$.

Meanwhile, as specified in Step-I, each accumulation step involves one column of matrix $[G_0]$ (or $[G_1]$), which is actually one complete column of the original matrix $[G]$ though only the left half side of the matrix is involved. This discovery inspires us to treat the parallel accumulations of (17) and (18) as one complete operation to facilitate further implementation.

In particular, the obtaining of $[-G_1-G_0]$ and $[G_1-G_0]$ are no longer treated as separate operations but rather the

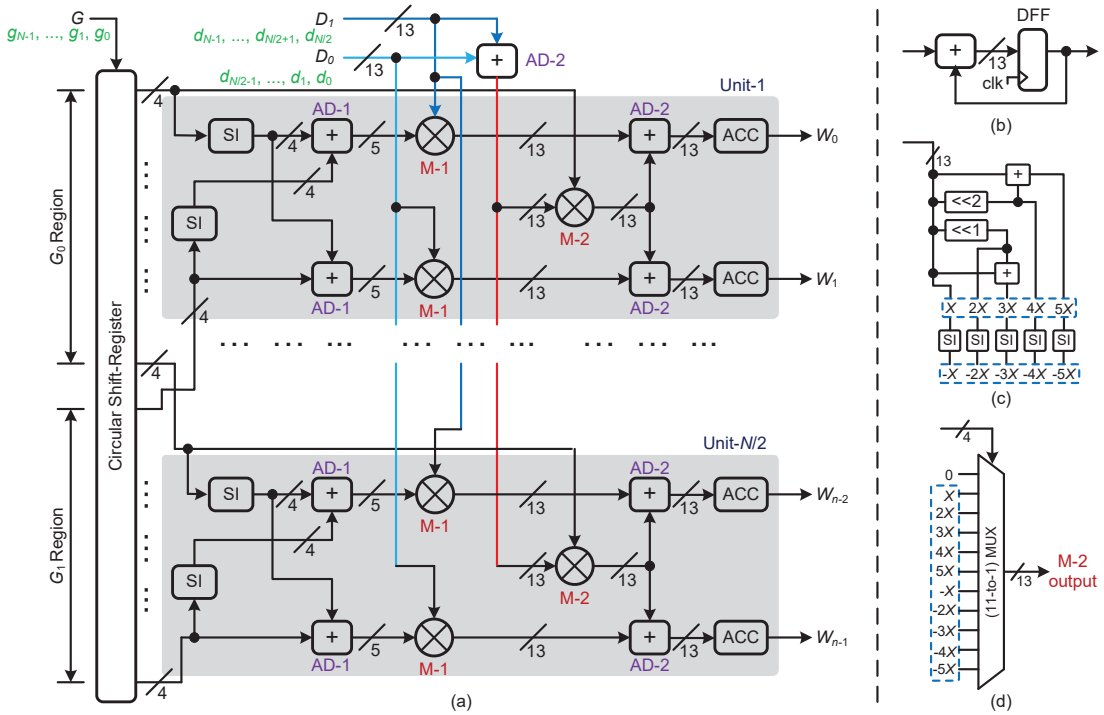

Fig. 1: (a) The proposed polynomial multiplication architecture based on Algorithm 1; SI: sign inverter. (b) The accumulator (ACC). (c) and (d) Design details for M-2, where (c) is the common factors shared by all the M-2 cells and (d) is the 11-to-1 MUX in the M-2 cell to produce the final output.

operation based on one complete column of the original matrix $[G]$. Then, the corresponding elements of $[-G_1 - G_0]$ and $[G_1 - G_0]$ will be produced through the element-wise based sign inverting and adding/subtracting operations (Section IV).

The Proposed TMVP-based Algorithm. Based on the above derivations, we can thus derive the proposed algorithm as:

Algorithm 1: Proposed TMVP based polynomial multiplication algorithm for KEM Saber

Input : G and D are integer polynomials. // the actual bit-width of the coefficients follow Notation 1.

Output: $W = GD \bmod (x^N + 1)$.

Initialization step

make ready the inputs G and D.

$[\overline{W_0}] = [\overline{W_1}] = [0]$; // $[\overline{W_0}]$ and $[\overline{W_1}]$ are $\frac{N}{2} \times 1$ matrices.

Main step

for $i = 1$ *to* $N/2$ **do**

  for $j = 1$ *to* $N/2$ **do**

    $[\overline{W_0}] = [\overline{W_0}] + [G_0]_{i,j}[D_0 + D_1]_j + [-G_1 - G_0]_{i,j}[D_1]_j$.

    $[\overline{W_1}] = [\overline{W_1}] + [G_0]_{i,j}[D_0 + D_1]_j + [G_1 - G_0]_{i,j}[D_0]_{j,1}$. // following (17)-(18)

  end

end

$[W_0] = [\overline{W_0}]$; $[W_1] = [\overline{W_1}]$.

Final step

deliver all the coefficients of output W;

which fully fulfills the objectives of the proposed mathematical derivation strategy.

IV. Proposed Polynomial Multiplication Hardware Architecture

The proposed polynomial multiplication hardware architecture is shown in Fig. 1, which is obtained through several algorithm-architecture co-implementation techniques.

Overall Description. The proposed architecture of Fig. 1 contains $N/2$ number of arithmetic units (highlighted as the gray box) to produce N coefficients of W. Besides that, a circular shift-register (CSR) is needed to load all the coefficients of polynomial G into the proper position to be further processed to produce the correct output to the $N/2$ arithmetic units. Meanwhile, the coefficients from two $N/2$-size matrix-vectors are also serially fed to the structure. The N output coefficients of W are delivered out in a parallel format. Note that all the input/output data are processed in the two's complement representation form. The details of the involved components, as well as related techniques, are listed below.

CSR. The internal structure of the CSR is shown in Fig. 2, where in total there are N number of registers and one MUX. In the loading stage, the selection of the MUX (sel-1) is set as '1' that the input from polynomial G are serially loaded into the registers. After that, the selection signal (sel-1) is switched to '0' that the values loaded in the registers are circularly shifted once per cycle. Note that a sign inverter (SI) is required to invert the sign of the bottom value to generate the correct output. The outputs of the registers are connected to the outside to be used as the correct values to constitute

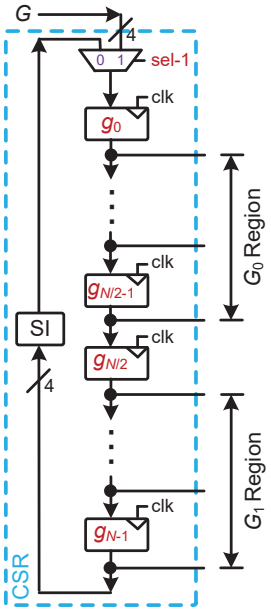

Fig. 2: The CSR (values in the registers are initial loadings).

the same column of matrices $[G_0]$ and $[G_1]$ (Algorithm 1), specified as the G_0 and G_1 regions, respectively, in Fig. 2.

Arithmetic Unit. The arithmetic unit, as highlighted in Fig. 1, is the major component for the proposed polynomial multiplication architecture. According to the procedures shown in Steps 5 and 6 of Algorithm 1, we can firstly consider two important operations involved there: the two corresponding elements (with sign inverted) from matrix $[G_0]$ and matrix $[G_1]$ need to be added together to be multiplied with one coefficient from $[D_1]$ (Step 5). In contrast, the same elements from matrix $[G_0]$ and matrix $[G_1]$ are subtracted, which can be seen as the addition with another element with sign inverted, and then multiplied with the related coefficient from $[D_0]$ (Step 6). These two operations can thus be transferred into the two SI cells, two 4-bit adders (AD-1), and two following multipliers (M-1) within one arithmetic unit, as shown in Fig. 1. Note that there is another pair of addition operations involved within Steps 5 and 6 of Algorithm 1, i.e., $[D_0+D_1]_j$. But for the sake of resource usage saving, we purposely arrange the elements from $[G_0]$ and $[G_1]$ be added in the parallel format (in $N/2$ arithmetic units) since the 4-bit adder has smaller area usage than the 13-bit one.

Meanwhile, to further save the resource usage, we follow the existing design style of [14] that all the M-1 and M-2 cells connecting with the same input from D_0 (or D_1 or their addition output) share the common factors. Consider that the coefficients generated from the sampler (input G) are in the range of [-5,5] (unified implementation including all three security levels of KEM Saber) and the output style of the adder (AD-2 for serial inputs D_0 and D_1), we can thus use a MUX-based lookup-table (LUT) method to realize M-2. This setup requires all the possible output values to be pre-computed and shared with the other parallel $N/2$ adders in the arithmetic units. Each M-2 cell contains one 11-to-1 MUX (the output of the AD-2 is used as the selection signal to the MUX) to execute the proper operation. As shown in Figs. 1(c) and (d), the values inside of the dotted blue box are the common factors to be shared among all the M-2 cells, and X is the value of the 13-bit input. For all the M-1 cells in Fig. 1, however, we use another MUX since the output of the adder (AD-1) now lies in the range of [-10,10]. We can use the 21-to-1 MUXes for all the M-2 cells, and all the 20 common factors are shared among all arithmetic units.

Lastly, according to Steps 5 and 6 of Algorithm 1, the elements of $[D_0]$ and $[D_1]$ are serially fed into the structure such that only one 13-bit adder is needed, as shown by the shared red line in Fig. 1. The output of each M-2 cell is added with the related outputs of two M-1 cells, respectively. Then, the outputs of the AD-2 are delivered to the accumulator (ACC) to produce the final output. As indicated in Section III, this kind of setup brings resource usage saving to the overall structure since an ACC has a larger area than an adder. As shown in Fig. 1(b), each ACC involves one adder and one register (DFF) connected in a feedback loop format such that the newly received input is added with the previous output of the adder to produce the output again for storing in the register. Note that one arithmetic unit produces two output coefficients, and the final output coefficients of W from all the arithmetic units are available in parallel.

Control Cell. A control cell is required to coordinate all the components in the structure of Fig. 1 function properly. The operational status of the architecture of Fig. 1 involves three stages, namely loading, computing, and delivering. The loading stage refers to the loading of input coefficients of G into the CSR. The computing stage denotes the computation of the required operations according to mainly Steps 5 and 6 of Algorithm 1 to produce the correct output. The delivering stage refers to the transferring output results into the memory.

Polynomial Multiplication Deploying Consideration. For the deploying of the proposed polynomial multiplication core, we have made several extra efforts and updates:

Data Input/Output Buffer. As shown in Algorithm 1, the proposed polynomial multiplier takes two inputs G and D, and produces one output W. Thus, the proposed architecture requires two input buffers and one output buffer. (a) Since each coefficient of input polynomial G is 4-bit in length and these coefficients are assumed to be sent from regular memory as 64-bit (regular in modern processor), the input buffer of G contains $N = 256$ cells (each cell contains four 1-bit registers). During the loading phase, the data in each cell is passed to the 16th cell ahead of the current location. In this way, the loading phase of the secret-key G takes only 16 cycles. Then, during the computation phase, each cell passes data to the next cell ahead to execute the necessary computation according to the nature of the for-loop in Algorithm 1. (b) The input buffer of D consists of four 16-bit cells, which are aligned up to receive one 64-bit word from external memory. In odd cycles, a new word is loaded into D's input buffer; while in even cycles, data in the bottom two cells are shifted into the top two cells. In this way, the core of the

TABLE II: Comparison of the Area-Time Complexities for the Proposed and Existing Hardware Implementations

design	LUT	FF	CLB	Fmax (MHz)	DSPs	latency[!]	delay (μs)	ADP[0]	ADPR[#]
[11][1]	17,406	5,069	2712	250	0	256	1.02	17,823.74	-10.75%
[12][2]	13,735	4,486	-	160	85	83	0.52	$*$	-
[23] (FIR) [3]	16,971	8,755	-	250	0	511	2.04	34,688.72	-115.54%
[23] (Fast.2)[3]	25,831	12,850	-	250	0	255	1.02	26,347.62	-63.71%
[23] (Fast.4)[3]	35,306	19,143	-	250	64	127	0.51	$*$	-
[24]	26,884	14,524	4,419	441	0	264	0.60	16,093.82	-
Proposed	30,047	7,680	4,834	344	0	128	0.37	11,117.39	30.92%

[!]: latency denotes to the number of clock cycles.
[0]: ADP refers to area-delay product, which is ADP=#LUT$\times$delay (since some of the existing designs do not report the CLB usage).
[#] :ADPR denotes the ADP reduction, where the design of [24] is used as a baseline to calculate the corresponding results.
[1]: The reported number of CLBs is obtained from the released source code.
[2]: This design is based on Karatsuba algorithm.
[3]: This design is based on filtering-based fast algorithm (Fast.4 has smaller latency but with larger resource usage).
$*$: These designs use large number of DSPs, which is difficult to calculate the actual ADP here. This is because one DSP typically can be seen as equivalent to 102.4 slices [25], which indicates that the designs of [12], [23] need at least 8,704 and 6,554 equivalent CLBs (for using DSPs).

proposed multiplier can read two coefficients of polynomial D from the top two cells in each cycle. The most significant three bits of each cell are abandoned since each coefficient of D is 13-bit in length. (c) The output buffer for polynomial W consists of 64 cells with each cell as 64-bit length. All output coefficients are loaded into the output buffer in one cycle when the result is ready. In the following 64 cycles, the data is shifted into the memory one cell by another. Each output cell contains only 52 1-bit registers in the actual implementation since each coefficient is coded in 13-bit.

Meanwhile, the control cell for the polynomial multiplication is also updated to coordinate with the input/output data transferring. Overall, The proposed architecture of Fig. 1 requires only $N/2$ cycles of accumulation to generate the output W, which benefited from the proposed TMVP-based algorithm. Besides that, the proposed polynomial multiplication is a unified structure that fits all security levels of Saber.

V. Complexity & Comparison

The proposed polynomial multiplication architecture (including the input/output buffer and control cell) is coded in VHDL with functionality verified. Finally, we have used Vivado 2020.2 to synthesize and implement it on the targeted AMD-Xilinx UltraScale+ XCZU9EG-2FFVB1156 FPGA device. Note that the implemented unified architecture fits well for all three security levels of KEM Saber with $N = 256$, and the detailed performance results are provided as follows.

Complexity. The area usage for the proposed polynomial multiplier is listed in Table II, where the maximum frequency reaches to 344MHz. One can notice that the polynomial multiplier occupies 30,047 LUTs and 7,680 FFs (4,834 CLBs). Meanwhile, the time-complexity of the proposed hardware accelerator, in terms of the number of computation cycles and related computational time, are also listed in Table II.

Comparison With The Existing Implementations. We have compared the area-time complexities of the proposed architecture with the existing implementations to demonstrate the efficiency of the proposed design strategy.

Comparison Consideration. Considering the facts that: (i) the proposed polynomial multiplication is a unified architecture for high-performance operations; (ii) the proposed polynomial multiplication is based on the TMVP fast algorithm; (iii) the polynomial multiplication is implemented with actual deploying consideration. We thus: (a) compare our proposed architecture with the existing high-performance hardware polynomial multipliers for Saber ([11], [12]); (b) also compare the high-performance hardware designs with practical setup on input/output processing [23], [24] ([26] and [14] do not have this setup and hence are not listed for comparison); (c) we do not include compact designs for KEM Saber (like [15]) due to the processing style difference.

Comparison Details. The detailed complexities of the proposed architectures are listed in Table II, along with the available implementations [11], [12], [23], [24].

As seen from Table II, when comparing with the existing KEM Saber polynomial multiplication implementations, the proposed accelerator has significantly outperformed the existing designs of [11], [12], [23], [24]. For instance, the proposed design has 30.92% less area-delay product (ADP) than the existing designs. As the proposed accelerator is a unified architecture, the comparison with the existing unified design of [24] reflects more precise information on the proposed architecture's superior performance. It is shown that the proposed accelerator has slightly larger area usage (because of the processing three matrix-vector products at the same time) but with a much shorter time. More detailedly, the proposed accelerator has 11.77% more LUTs and 47.12% less FFs (overall 9.4% larger CLBs) than the existing unified design of [24] while involves 38.33% less latency time. Hence, the proposed accelerator involves significantly smaller area-time complexities than the one of [24]. The similar situation happens to the comparison with [11] and two designs of [23].

Apart from that, when considering the comparison with [12] and one design of [23] (see Table II), the proposed polynomial multiplier still obtains much small resource usage while maintains very efficient time-complexity. This is because the existing architectures of [12], [23] use very large number of DSPs, which equivalent to at least 8,704 and 6,554 equivalent CLBs (just for DSPs) based on the estimation standard proposed in [25].

Overall, due to the benefits brought by the proposed TMVP design strategy, the proposed polynomial multiplication un-

doubtedly has more balanced performance and is more suitable for practical application than the existing ones.

Discussion. The proposed TMVP-based polynomial multiplier features: (i) shorter computational latency; (ii) reduced complexity and resource usage due to the TMVP strategy (three $\frac{N}{2}$-size matrix-vector products); (iii) simple architecture with efficient implementation. As this is the first TMVP-based hardware polynomial multiplication for Saber, we expect this strategy can be extended for other PQC implementation.

Other Works and Future Research. There also exist important lattice-based PQC designs includes the Ring-LWE-based designs of [27], [28]. Though these designs have different focuses, they represent the major advance in the field. Future work will focus on side-channel attacks and countermeasures.

VI. Conclusion

In this paper, we propose a novel TMVP-based polynomial multiplication for KEM Saber. Firstly, we have formulated the polynomial multiplication of KEM Saber into the proposed TMVP-based algorithm. Then, we have transferred the proposed TMVP-based algorithm into the desired polynomial multiplication architecture. Finally, detailed implementation results and complexity analysis have shown that the proposed polynomial multiplication structure has significantly better area-time complexities than the state-of-the-art solutions. To the authors' best knowledge, this is the first report on the hardware implementation of polynomial multiplication of Saber based on TMVP. The research outcome of this work is expected to generate significant impacts on efficient implementation for other PQC schemes.

VII. Acknowledgement

The work of J. Xie was supported by NIST-60NANB20D20 and in part by NSF SaTC-2020625.

References

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994.

[2] J. Xie, K. Basu, K. Gaj, and U. Guin, "Special session: The recent advance in hardware implementation of post-quantum cryptography," in *2020 IEEE 38th VLSI Test Symposium (VTS)*, pp. 1–10, IEEE, 2020.

[3] *Post quantum cryptography round 3 submissions. https://csrc.nist.gov/projects/post-quantumcryptography/round-3-submissions.*, 2020.

[4] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[5] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 1–23, Springer, 2010.

[6] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 719–737, Springer, 2012.

[7] J.-P. D'Anvers, A. Karmakar, S. S. Roy, F. Vercauteren, J. Mera, M. Beirendonck, and A. Basso, "SABER: Mod-LWR based KEM (Round 3 Submission),"

[8] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *International Conference on Cryptology in Africa*, pp. 282–305, Springer, 2018.

[9] J. M. B. Mera, F. Turan, A. Karmakar, S. S. Roy, and I. Verbauwhede, "Compact domain-specific co-processor for accelerating module lattice-based KEM," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2020.

[10] V. B. Dang, F. Farahmand, M. Andrzejczak, and K. Gaj, "Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, pp. 206–214, IEEE, 2019.

[11] S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 443–466, 2020.

[12] Y. Zhu, M. Zhu, B. Yang, W. Zhu, C. Deng, C. Chen, S. Wei, and L. Liu, "LWRpro: An energy-efficient configurable crypto-processor for Module-LWR," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 3, pp. 1146–1159, 2021.

[13] E. Carter, P. He, and J. Xie, "High-performance polynomial multiplication hardware accelerators for kem saber and ntru," *Cryptology ePrint Archive*, 2022.

[14] A. Basso and S. S. Roy, "Optimized polynomial multiplier architectures for post-quantum KEM Saber," *Design Automation Conference (DAC)'21*, 2021.

[15] P. He, C.-Y. Lee, and J. Xie, "Compact coprocessor for KEM Saber: Novel scalable matrix originated processing," *The NIST Third Standardization Conference*, pp. 1–16, 2021.

[16] V. B. Dang, K. Mohajerani, and K. Gaj, "High-speed hardware architectures and fair FPGA benchmarking of CRYSTALS-Kyber, NTRU, and Saber," *The NIST Third Standardization Conference*, pp. 1–48, 2021.

[17] H. Fan and M. A. Hasan, "A new approach to subquadratic space complexity parallel multipliers for extended binary fields," *IEEE Transactions on Computers*, vol. 56, no. 2, pp. 224–233, 2007.

[18] M. A. Hasan, N. Meloni, A. H. Namin, and C. Negre, "Block recombination approach for subquadratic space complexity binary field multiplication based on Toeplitz matrix-vector product," *IEEE Transactions on Computers*, vol. 61, no. 2, pp. 151–163, 2010.

[19] C.-Y. Lee and P. K. Meher, "Area-efficient subquadratic space-complexity digit-serial multiplier for type-ii optimal normal basis of $GF(2^m)$ using symmetric TMVP and block recombination techniques," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 12, pp. 2846–2855, 2015.

[20] C.-Y. Lee and J. Xie, "High capability and low-complexity: Novel fault detection scheme for finite field multipliers over $GF(2^m)$ based on MSPB," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 21–30, IEEE, 2019.

[21] J.-S. Pan and et al., "Novel systolization of subquadratic space complexity multipliers based on Toeplitz matrix–vector product approach," *IEEE TVLSI Systems*, vol. 27, no. 7, pp. 1614–1622, 2019.

[22] I. K. Paksoy and M. Cenk, "TMVP-based multiplication for polynomial quotient rings and application to Saber on ARM Cortex-M4," *https://eprint.iacr.org/2020/1302*, 2020.

[23] W. Tan, A. Wang, Y. Lao, X. Zhang, and K. K. Parhi, "Low-latency vlsi architectures for modular polynomial multiplication via fast filtering and applications to lattice-based cryptography," *arXiv preprint arXiv:2110.12127*, 2021.

[24] P. He and et al., "HPMA-Saber: High-performance polynomial multiplication accelerator for kem saber," *International Conference on Computer Design (ICCD)*, pp. 1–4, 2022.

[25] Y. Zhang and et al., "An efficient and parallel r-lwe cryptoprocessor," *IEEE TCAS-II*, vol. 67, no. 5, pp. 886–890, 2020.

[26] J. Xie and et al., "Crop: Fpga implementation of high-performance polynomial multiplication in saber kem based on novel cyclic-row oriented processing strategy," in *IEEE ICCD*, pp. 130–137, IEEE, 2021.

[27] H. Nejatollahi, S. Shahhosseini, R. Cammarota, and N. Dutt, "Exploring energy efficient quantum-resistant signal processing using array processors," in *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1539–1543, IEEE, 2020.

[28] B. J. Lucas and et al., "Lightweight hardware implementation of binary ring-lwe pqc accelerator," *IEEE Computer Architecture Letters*, vol. 21, no. 1, pp. 17–20, 2022.