Efficient Hardware Arithmetic for Inverted Binary Ring-LWE Based Post-Quantum Cryptography

José L. Imaña, Pengzhou He, Tianyou Bao, Yazheng Tu, and Jiafeng Xie, *Senior Member, IEEE*

***Abstract*— *Ring learning-with-errors* (RLWE)-based encryption scheme is a lattice-based cryptographic algorithm that constitutes one of the most promising candidates for Post-Quantum Cryptography (PQC) standardization due to its efficient implementation and low computational complexity. *Binary Ring*-LWE (BRLWE) is a new optimized variant of RLWE, which achieves smaller computational complexity and higher efficient hardware implementations. In this paper, two efficient architectures based on *Linear-Feedback Shift Register* (LFSR) for the arithmetic used in *Inverted Binary Ring*-LWE (*Inv*BRLWE)-based encryption scheme are presented, namely the operation of $A \cdot B + C$ over the polynomial ring $\mathbb{Z}_q/(x^n+1)$. The first architecture optimizes the resource usage for major computation and has a novel input processing setup to speed up the overall processing latency with minimized input loading cycles. The second architecture deploys an innovative serial-in serial-out processing format to reduce the involved area usage further yet maintains a regular input loading time-complexity. Experimental results show that the architectures presented here improve the complexities obtained by competing schemes found in the literature, e.g., involving 71.23% less area-delay product than recent designs. Both architectures are highly efficient in terms of area-time complexities and can be extended for deploying in different lightweight application environments.**

***Index Terms*— Binary ring-LWE, hardware design, lattice-based, LFSR, post-quantum cryptography, polynomial arithmetic.**

I. Introduction

THE rapid advancement in quantum computing represents an important threat to modern Public-Key Cryptography (PKC) that the most widely used public-key cryptosystems, such as RSA and Elliptic Curve Cryptography (ECC), could be easily broken by powerful quantum computers executing Shor's algorithm [1]. For this reason, Post-Quantum Cryptography (PQC) [2], representing cryptosystems that can resist both classical and quantum attacks, has become an essential research topic recently. The National Institute of Standards and Technology (NIST) has initiated a PQC standardization process, where the most promising cryptosystems are mainly classified as *lattice-based* [3], *code-based* [4], *hash-based* [5], *isogeny-based* [6], and *multivariate-quadratic cryptography* [7]. Apart from the security proof, efficient hardware implementations for these candidates are also needed [8]–[10].

Overall, *lattice-based* cryptography is one of the most promising candidates due to its strong security proof, low complexity, and efficient implementation [11]. The lattice-based cryptosystem relies on the hardness of the *learning-with-errors* (LWE) problem [12]. and have shown resistance to several types of attacks [13]. *Ring*-LWE (RLWE) [14] involves arithmetic operations over a polynomial ring and has smaller key sizes than the original LWE. A relatively new variant of *Ring*-LWE was proposed in [22] that uses binary errors instead of the regular Gaussian one, resulting in much smaller key sizes [15]–[21]. This optimized variant was named as *Binary Ring*-LWE (BRLWE), from which efficient public-key encryption scheme was built [22].

A. Existing Works

Since the introduction of BRLWE in [22], a number of hardware implementations have been released for this encryption scheme: the first hardware design was released in [23]; a pair of high-speed and low-speed hardware design were then given in [24]; a high-speed structure was presented in [25] but with an incomplete hardware setup (e.g., lacking proper sign control). Another high-performance hardware BRLWE-based PQC was recently reported in [27]. A compact design was presented in [26]. Two new high-speed architectures were released in [27] and [28], respectively. A new low-complexity structure was given in [29]. Another pair of low-speed and high-speed architectures were very recently reported in [31]. Other works include the fault detection architecture of [10] (based on [24]) and fault analysis (software based) of [30]. These reports are the main works in the field.

Overall, the existing works (mostly high-speed architectures) can be categorized into mainly two types, in terms of their processing design styles: (i) parallel-in parallel-out (one or more inputs/outputs are directly connected to the main structure in a parallel format [23]–[25], [27]); (ii) serial-in serial-out (where the key inputs/outputs are

Manuscript received January 15, 2022; revised April 1, 2022; accepted April 18, 2022. The work of José L. Imaña was supported in part by the Spanish Government Ministerio de Economia y Competitividad (MINECO) under Grant RTI2018-093684-B-I00 and in part by the Comunidad de Madrid under Grant S2018/TCS-4423. The work of Jiafeng Xie was supported by the National Science Foundation (NSF) Award under Grant SaTC-2020625 and Grant NIST-60NANB20D203. This article was recommended by Associate Editor W. Liu. *(José L. Imaña and Pengzhou He contributed equally to this work.) (Corresponding author: José L. Imaña.)*

José L. Imaña is with the Department of Computer Architecture and Automation, Faculty of Physics, Complutense University of Madrid, 28040 Madrid, Spain (e-mail: jluimana@ucm.es).

Pengzhou He, Tianyou Bao, Yazheng Tu, and Jiafeng Xie are with the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA 19085 USA (e-mail: phe@villanova.edu; tbao@villanova.edu; ytu1@villanova.edu; jiafeng.xie@villanova.edu).

Digital Object Identifier 10.1109/TCSI.2022.3169471

serially loaded-in/delivered-out from the main structure [28]). These high-speed designs, however, have not fully explored the efficient hardware implementation of the BRLWE-based encryption scheme. For instance, most of the existing parallel-in parallel-out hardware structures do not fully optimize the involved area usage with the processing speed. While the existing serial-in serial-out structure still involves extra resources for structural operation, e.g., the recently released one of [28] needs an extra sign control unit for accurate operation.

Based on the above considerations, in this paper, we propose two novel hardware architectures based on *Linear-Feedback Shift Register* (LFSR) for the major arithmetic used in the *Inverted Binary Ring*-LWE (*Inv*BRLWE)-based encryption scheme (an inverted variant of BRLWE given in [24]), i.e., the operation $A \cdot B + C$ over ring $\mathbb{Z}_q[x]/(x^n+1)$, where polynomial coefficients of A and C are in $\mathbb{Z}_q$ and coefficients of B are binary values [22]. These two new architectures exhibit great capability for high-speed and low-complexity applications, respectively, as listed below (major contributions):

- The first LFSR-based architecture optimizes the resource usage and the processing speed. This structure speeds up the input loading with a novel parallel-in serial-out setup.
- The second architecture can process the arithmetic operation in a serial-in serial-out format to further reduce area-complexity yet with regular input loading time.
- The arithmetic architectures presented here improve the complexities obtained by competing schemes found in the literature, as demonstrated by the following implementation based comparison and discussion.

The rest of the paper is organized as follows. Section II provides basic mathematical concepts. Two new LFSR-based architectures for $A \cdot B + C$ in $\mathbb{Z}_q[x]/(x^n+1)$ are introduced in Section III. Comparisons with the existing designs based on field-programmable gate array (FPGA) results are presented in Section IV. Finally, conclusions are given in Section V.

II. Background

In this section, the main mathematical concepts used in this paper (*Inv*BRLWE-based scheme and LFSR) are given.

A. Polynomial Rings

Let q be an integer $q \in \mathbb{Z}$, then the finite ring modulo q is defined as $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z} = \{0, 1, \ldots, q-1\}$. The set of polynomials in the indeterminate x with coefficients in $\mathbb{Z}_q$ is represented as $\mathbb{Z}_q[x]$. The ring of polynomials with coefficients in $\mathbb{Z}_q$ is $\mathcal{R}_q = \mathbb{Z}_q[x]/f(x)$, where $f(x) \in \mathbb{Z}_q[x]$ is the modulo of the ring. If $f(x)$ is a polynomial of degree n, then an element in $\mathcal{R}_q$ is a polynomial of degree (n-1) with n coefficients in $\mathbb{Z}_q$, i.e. n integer coefficients modulo q.

B. Inverted Binary Ring-LWE (InvBRLWE)

Lattice-based cryptographic constructions are based on the hardness of the *lattice* problems [14]. A lattice-based cryptosystem, relies on the hardness of the LWE problem, was given in [12]. In LWE, a secret $s \in \mathbb{Z}_q^n = \{(x_0, \ldots, x_n)\}$, with $x_i \in \mathbb{Z}_q$, has to be found in several pairs (a, b_i) with $b_i = a \cdot s + e_i$, where $a \in \mathbb{Z}_q^n$ is known and uniformly random and $e_i \in \mathbb{Z}_q$ are error vectors according to a Gaussian distribution over $\mathbb{Z}_q$. The Ring-LWE based scheme [14] uses ideal lattices in LWE and has smaller key sizes than the original LWE-based scheme. Ring-LWE involves arithmetic operations over the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$. *Addition* and *subtraction* in this ring correspond with a coefficient-wise addition and subtraction modulo q, respectively. *Multiplication* involves a polynomial multiplication that generates an $(2n-2)$ degree polynomial followed by a reduction modulo $f(x) = x^n + 1$, resulting in an $(n-1)$ degree polynomial. The coefficients of this resulting polynomial should also be reduced modulo q. *Binary Ring*-LWE (BRLWE) is a relatively new variant of *Ring*-LWE [22] with smaller key sizes and efficient implementations. BRLWE-based scheme consists of three steps:

- *Key generation*: Let $p = r_1 - a \cdot r_2$, where $r_1, r_2 \in \{0,1\}^n$ are random binary vectors and $a \in \mathcal{R}_q$ is publicly known. The *private key* is r_2 and $p \in \mathcal{R}_q$ is the *public key*.
- *Encryption*: The input message $m_e \in \{0,1\}^n$ is encoded into a polynomial $\tilde{m} \in \mathcal{R}_q$ by multiplying each coefficient by $q/2$. The ciphertext involves operations $c_1 = a \cdot e_1 + e_2$ and $c_2 = p \cdot e_1 + e_3 + \tilde{m}$, where $c_1, c_2 \in \mathcal{R}_q$ and $e_1, e_2, e_3 \in \{0,1\}^n$ are random binary vectors.
- *Decryption*: The original message m_e is decrypted by computing $c = c_1 \cdot r_2 + c_2$ (with $c \in \mathcal{R}_q$), and applying it a *threshold decoder function* that $m_i =$'1' if c_i is within the range $(q/4, 3q/4)$ and $m_i =$'0' otherwise.

Security of the BRLWE-based PQC. The BRLWE-based encryption scheme is based on the average-case hardness of the BRLWE problem [22]. The work of [32] showed that this scheme achieves 73-bits and 140-bits quantum security for the parameters of $(n, q) = (256, 256)$ and $(n, q) = (512, 256)$, respectively, which fits well typical lightweight applications.

Inverted Binary Ring-LWE (*Inv*BRLWE)-based encryption scheme [24] originates from a recent variant of BRLWE where the coefficients of the polynomials in $\mathcal{R}_q$ are selected from $\mathbb{Z}_q$ with range $\{-\lfloor q/2 \rfloor, \ldots, \lfloor q/2 \rfloor - 1\}$. This inverted range matches with the *two's complement* representation [24], so the polynomial coefficients in $\mathcal{R}_q$ can be computed without any extra reduction [24]. Each coefficient of the input message needs to be multiplied by $(-q/2)$. In this paper, the targeted arithmetic operation is based on the *Inv*BRLWE-based PQC.

C. Linear-Feedback Shift Register (LFSR)

LFSR is a shift-register whose feedback value is a linear function of its previous state. We may use the finite field arithmetic over $GF(2^m)$ [33] to illustrate the details of LFSR. The product $A' \cdot x \bmod f'(x)$ (x is the root of the characteristic polynomial $f'(x)$) can be performed using an LFSR with m 1-bit registers, where the registers are initially loaded with the coordinates of the element A' and the coefficients f_i, with $i = 1 \ldots m-1$. After m clock cycles, the registers store the coefficients of the product [34]. A new LFSR-based multiplier over $GF(2^m)$ was given in [35], as shown in Fig. 1. Note that $\oplus$ refers to an XOR gate, $\boxtimes$ denotes an AND gate, and $\square$ stands for a 1-bit register. After m clock cycles, the registers c'_i

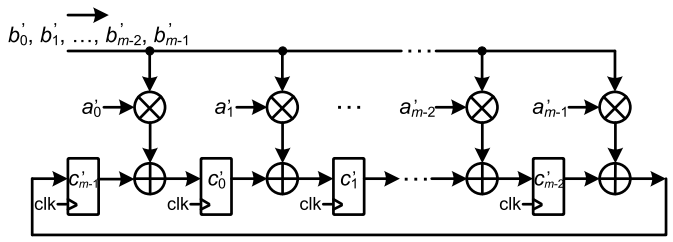


Fig. 1. The LFSR-based architecture for implementation of $A' \cdot B'$ mod $(x^m + 1)$ over $GF(2^m)$.

$(i = 0 \ldots m-1)$ of Fig. 1 store the corresponding binary coefficients of the $GF(2^m)$ multiplication modulo $f'(x) = x^m + 1$. An LFSR-based design for PQC with an efficient multiplier was given in [36] for *NTRUEncrypt* scheme. Another efficient multiplier for NTRU prime was given in [37]. The structure of [28] can also be seen as a new variant of LFSR-based design.

III. LFSR-Based Architectures for $A \cdot B + C$ in $\mathcal{R}_q$ for *Inv*BRLWE-Based Encryption Scheme

A. Arithmetic Consideration and Derivation

It can be observed from Section II that the arithmetic operation involved in the *Inv*BRLWE-based encryption scheme is $A \cdot B$ mod $(x^n + 1) + C$, where A and C are polynomials with n coefficients in $\mathbb{Z}_q$ and B is a binary polynomial of degree $(n-1)$. The product $A \cdot B$ mod $(x^n + 1)$ involves a polynomial multiplication that generates an $(2n-2)$ degree polynomial that must be followed by a reduction modulo $f(x) = x^n + 1$, resulting in a $(n-1)$ degree polynomial with coefficients in $\mathbb{Z}_q$. As B is a binary polynomial, no reduction modulo q of the resulting coefficients is needed. The addition of this product with polynomial C corresponds with the coefficient-wise addition modulo q. As mentioned earlier, the range of $\mathbb{Z}_q$ coefficients matches the *two's complement* representation range for an integer $q = 2^k$ with $k = \log_2 q$ bits. Therefore, in the *Inv*BRLWE-based PQC, the coefficients of the polynomials can be represented with two's complement notation and modular addition can be computed without any extra reduction [24]. Meanwhile, the involved polynomial multiplication $A \cdot B$, where $A = \sum_{i=0}^{n-1} a_i x^i$ and $B = \sum_{i=0}^{n-1} b_i x^i$ with $a_i \in \mathbb{Z}_q$ and $b_i \in \{0, 1\}$, generates a $(2n-2)$ degree polynomial. This polynomial must be reduced through modulo $f(x) = x^n + 1$, in such a way that $x^n \equiv -1$, $x^{n+1} = -x$, …, and $x^{2n-2} = -x^{n-2}$. For example, for the product $A \cdot B$ mod $(x^6 + 1) = (a_5 x^5 + \ldots + a_1 x + a_0) \cdot (b_5 x^5 + \ldots + b_1 x + b_0)$ mod $(x^6 + 1)$, it can be checked that the coefficients associated with the x^i weights, with $i = 0 \ldots 5$, are given in Table I.

1) Two Important Properties: The polynomial B used in the *Inv*BRLWE-based encryption scheme has binary coefficients, so the products $a_i b_j$ are just integers represented by the *two's complement* format. Therefore, all arithmetic operations shown in Table I can be performed under two's complement (C_2') representation. As subtraction equals addition of its opposite, we can thus represent the opposite of a number Y as $C_2'(Y) = 2^k - Y$ (assume we are using k-bits under the two's complement representation). Then, we can have the following two important computational properties.

TABLE I

Computation of $A \cdot B$ Mod $(x^6 + 1)$

x^0	$a_0b_0 - a_5b_1 - a_4b_2 - a_3b_3 - a_2b_4 - a_1b_5$
x^1	$a_1b_0 + a_0b_1 - a_5b_2 - a_4b_3 - a_3b_4 - a_2b_5$
x^2	$a_2b_0 + a_1b_1 + a_0b_2 - a_5b_3 - a_4b_4 - a_3b_5$
x^3	$a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3 - a_5b_4 - a_4b_5$
x^4	$a_4b_0 + a_3b_1 + a_2b_2 + a_1b_3 + a_0b_4 - a_5b_5$
x^5	$a_5b_0 + a_4b_1 + a_3b_2 + a_2b_3 + a_1b_4 + a_0b_5$

First, the subtraction of two integers can be computed as $Z - Y = Z + C_2'(Y)$. It also holds that $-Z - Y = C'2(Z) + C_2'(Y) = C_2'(Z + Y) = -(Z + Y)$, which can be easily proven as: $C_2'(Z) + C_2'(Y) = 2^k - Z + 2^k - Y = 2^k + 2^k - (Z + Y) = 2^k - (Z + Y) = C_2'(Z + Y)$, i.e., one of the terms 2^k is discarded because it represents a '1' in the position with weight 2^k and therefore exceeds the number of k bits we use.

Second, we have $C_2'(Y) = C_1'(Y) + 1$, where $C_1'(Y)$ is the *one's complement* of Y (that is simply computed by the bit-wise NOT of the k bits of Y).

The above two properties can be used to compute the arithmetic expressions of Table I by merely additions (subtractions are performed by the additions, under the two's complement representation). For example, coefficient x^2 in Table I is computed as $a_2b_0 + a_1b_1 + a_0b_2 - a_5b_3 - a_4b_4 - a_3b_5 = a_2b_0 + a_1b_1 + a_0b_2 + C_2'(a_5b_3) + C_2'(a_4b_4) + C_2'(a_3b_5) = a_2b_0 + a_1b_1 + a_0b_2 + C_2'(a_5b_3 + a_4b_4 + a_3b_5)$. Note that in the rest of the paper, the two's complement of an integer Y is denoted as $\widehat{Y} = C_2'(Y)$.

2) Extension to the Final Algorithm: The above mentioned two important properties can be used to derive the final algorithm for the major arithmetic operation involved with the *Inv*BRLWE-based scheme (i,e., $AB + C$). Let us define again $C = \sum_{i=0}^{n-1} c_i x^i$ and $W = AB + C = \sum_{i=0}^{n-1} w_i x^i$ (for $c_i,\ w_i \in \mathbb{Z}_q$), we can have

$$\begin{aligned} W &= AB \bmod (x^n + 1) + C \\ &= A(b_0 + \cdots + b_{n-1}x^{n-1}) \bmod (x^n + 1) + C \\ &= [Ab_0 + \cdots + Ax^{n-1}b_{n-1}] \bmod (x^n + 1) + C, \quad (1) \end{aligned}$$

where we can further have $(x^n \equiv -1)$

$$\begin{aligned} Ax &= -a_{n-1} + a_0 x + a_1 x^2 + \cdots + a_{n-2}x^{n-1}, \\ Ax^2 &= -a_{n-2} - a_{n-1}x + a_0 x^2 + \cdots + a_{n-3}x^{n-1}, \\ &\ldots \ \ldots \ \ldots \\ Ax^{n-1} &= -a_1 - a_2 x - a_3 x^2 - \cdots + a_0 x^{n-1}, \quad (2) \end{aligned}$$

which can be re-substituted into (1) to have

$$\begin{aligned} w_0 &= a_0b_0 - a_{n-1}b_1 - \cdots - a_1b_{n-1} + c_0, \\ w_1 &= a_1b_0 + a_0b_1 - \cdots - a_2b_{n-1} + c_1, \\ &\ldots \ \ldots \ \ldots \\ w_{n-1} &= a_{n-1}b_0 + a_{n-2}b_1 + \cdots + a_0b_{n-1} + c_{n-1}, \quad (3) \end{aligned}$$

from where we can have Algorithm 1, where $[Ax^i]_j$ denotes the jth coefficient of polynomial Ax^i, e.g., $[Ax]_0 = -a_{n-1}$

Algorithm 1 Algorithmic Operation for $AB + C$ (*Inv*BRLWE-Based Encryption Scheme)

Input : A, C, and W are integer polynomials (coefficients $\in \mathbb{Z}_q$); B is binary polynomial.
Output: $W = AB \bmod (x^n + 1) + C$.
Initialization step
1 Make ready the inputs A, B, and C;
2 $\overline{W} = \sum_{i=0}^{n-1} \overline{w}_i x^i$; // executed in serial or parallel
Main step
3 **for** $i = 0$ *to* $n - 1$ **do**
4 | $\overline{w_i} = C_2'(c_i)$;
5 **end**
6 **for** $j = 0$ *to* $n - 1$ **do**
7 | **for** $i = 0$ *to* $n - 1$ **do**
8 | | $\overline{w}_j = \overline{w}_j + [Ax^i]_j b_i$; // following (3)
9 | **end**
10 **end**
11 $W = \overline{W}$;
Final step
12 Deliver all the coefficients of output W serially;

(see (2)). Note Algorithm 1 is executed with the combination of the highlighted two important properties.

3) Advantages of the Proposed Algorithmic Operation Over the Existing Ones: Comparing with the existing algorithmic operations for *Inv*BRLWE/BRLWE-based encryption scheme, i.e., the recent ones of [25], [27], [28], [31], Algorithm 1 has the following advantages. (i) The proposed algorithmic operation allows us to design high-speed structures with options of different input processing styles, while the existing algorithms mostly only fit for one type of design. (ii) The proposed algorithmic operation combines the mentioned two important properties that the required resource usage is minimized. For instance, the sign inversion brought by modular operation is executed smoothly without extra resource usage, while the existing algorithms, especially the recent one of [28], requires an additional sign control shift-register to carry out related operations. Meanwhile, the proposed Algorithm 1 allows the input loading and output delivery be executed by the same intermediate variable, which is more efficient than the algorithmic operation proposed in [31]. Note that the designs of [23], [24] did not report their algorithms and hence we do not discuss them here.

B. Proposed Architecture-I: Mostly Parallel-In Serial-Out

1) Definition: We define that a structure as mostly parallel-in serial-out when the coefficients of the major polynomials are fed in a parallel format (except that the coefficients of the binary polynomial are fed to the structure in serial). Meanwhile, the output results are serially delivered out.

Following this definition, we can have the proposed architecture-I for the major arithmetic operation ($W = A \cdot B \bmod (x^n + 1) + C$) involved within the *Inv*BRLWE-based encryption scheme, as shown in Fig. 2. **Note that** this proposed architecture is not a direct deploying of the LFSR-based architecture shown in Fig. 1, but with novel design features as described below, especially when considering the algorithmic operation of Algorithm 1 and the two important properties.

The proposed new LFSR-architecture, as given in Fig. 2, has used k-bit registers, k-bit adders, 2:1 k-bit multiplexers, and AND gates ($k = \log_2 q$) to perform the involved computation process. Connecting Algorithm 1 (Line 2), in order to perform the addition of the coefficients C with the corresponding coefficients of the product $A \cdot B \bmod (x^n + 1)$ (Line 5 of Algorithm 1), the coefficients c_i of polynomial C are firstly initialized into the k-bit registers (those will store w_i, for $i = 0 \ldots n - 1$), respectively. However, as the proposed architecture given in Fig. 2 includes the left *subtractor* in two's complement (also mentioned below), then the two's complement coefficients $C_2'(c_i) = \widehat{c_i}$ must be stored in registers (storing w_i) such that these values are again in the two's complement format with correct c_i after the right shifts. In order to initialize these values for the architecture given in Fig. 2, in total n number of 2:1 k-bit multiplexers with a control signal $init$ are needed. In this way, if $init =' 1'$ then the coefficients of C ($\widehat{c_i}$) are loaded into these w_i registers. When $init =' 0'$, then the computation of $W = A \cdot B \bmod (x^n + 1) + C$ starts to be carried out and the coefficients stored in each register will add with the corresponding result produced from those n parallel AND cells to meet the accumulation requirement (Line 5 of Algorithm 1), as demonstrated in Fig. 2 and described below.

First of all, the process of obtaining product of $[Ax^i]_j b_i$, with $a_i \in \mathbb{Z}_q$ and $b_i \in \{0, 1\}$, is carried out in Fig. 2 by k AND gates in parallel (in total nk AND gates when counting all products together), in such a way that if $b_j =$ '1' then $a_i b_j = a_i$ and if $b_j =$ '0' then the product is $a_i b_j =$ "0...00" (k bits). After that, the accumulation of these products $[Ax^i]_j b_i$ along with the related coefficients of C, as specified in Line 5 of Algorithm 1, can be mapped into a connected accumulation loop where the results of these products/multiplications are added with the values stored in the k-bit registers (will store w_i, $i = 0 \ldots n-1$) using k-bit adders, as shown in Fig. 2. Note that all the adders except the one on the most left in Fig. 2 have a $carry_{in}$ (*input carry*) equal to '0' in order to perform the required *addition*. However, the first adder (most left one) has to set $carry_{in} =$ '1', and one of its input operands is the inverted (bit-wise NOT) value delivered from the far-left register (the one will store w_{n-1}). Therefore, this most left adder performs the addition of $a_0 b_i + C_1'(Rw_{n-1}) +$ '1' $= a_0 b_i + C_2'(Rw_{n-1}) = a_0 b_i - Rw_{n-1}$ (Rw_{n-1} is the value stored in the far-right register), i.e., the exact *subtraction* of integers represented in *two's complement* format (according to (3)). After n cycles' accumulation, the corresponding results, namely the n coefficients of $W = AB \bmod (x^n + 1) + C$, will be stored in respective registers to be delivered out in a serial format (see the example below), i.e., from w_{n-1} to w_0. Note that during the output delivering process, the output of each AND cell will be zero such that no extra values will be accumulated during this process.

Of course, a n-length k-bit shift-register for A and a n-length 1-bit shift-register for B are needed for the loading/processing of the coefficients of input polynomials A and B, respectively, in the practical implementation process. While the delivery of

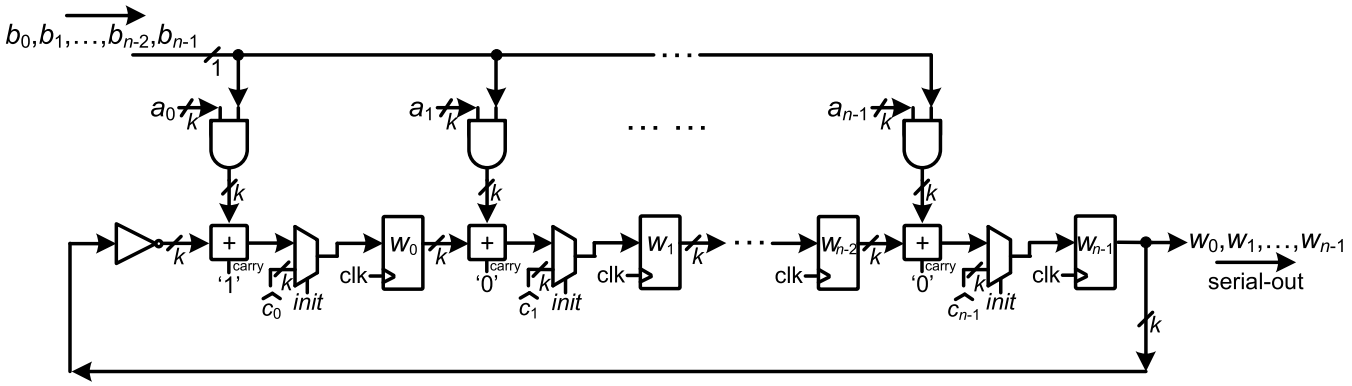


Fig. 2. Proposed new LFSR-based architecture-I for $A \cdot B$ mod (x^n+1) + C (*Inv*BRLWE-based encryption scheme), where the values in the registers (w_{n-1}, w_0, $\cdots$, w_{n-2}) are the final accumulated results to be delivered out.

TABLE II

COMPUTATION DETAILS OF $A \cdot B$ MOD (x^6+1) + C BASED ON THE NEW ARCHITECTURE OF FIG. 2

Cycle	Serial	Register-(for w_5)*	Register-(for w_0)*	Register-(for w_1)*	Register-(for w_2)*	Register-(for w_3)*	Register-(for w_4)*
init.	$-$	$\widehat{c_5}$	$\widehat{c_0}$	$\widehat{c_1}$	$\widehat{c_2}$	$\widehat{c_3}$	$\widehat{c_4}$
t_1	b_5	$a_5b_5+\widehat{c_4}$	$a_0b_5+c_5$	$a_1b_5+\widehat{c_0}$	$a_2b_5+\widehat{c_1}$	$a_3b_5+\widehat{c_2}$	$a_4b_5+\widehat{c_3}$
t_2	b_4	$a_4b_5+a_5b_4+\widehat{c_3}$	$\widehat{a_5b_5}+a_0b_4+c_4$	$a_0b_5+a_1b_4+c_5$	$a_1b_5+a_2b_4+\widehat{c_0}$	$a_2b_5+a_3b_4+\widehat{c_1}$	$a_3b_5+a_4b_4+\widehat{c_2}$
t_3	b_3	$a_3b_5+a_4b_4+$ $a_5b_3+\widehat{c_2}$	$\widehat{a_4b_5}+\widehat{a_5b_4}+$ $a_0b_3+c_3$	$\widehat{a_5b_5}+a_0b_4+$ $a_1b_3+c_4$	$a_0b_5+a_1b_4+$ $a_2b_3+c_5$	$a_1b_5+a_2b_4+$ $a_3b_3+\widehat{c_0}$	$a_2b_5+a_3b_4+$ $a_4b_3+\widehat{c_1}$
t_4	b_2	$a_2b_5+a_3b_4+$ $a_4b_3+a_5b_2+\widehat{c_1}$	$\widehat{a_3b_5}+\widehat{a_4b_4}+$ $\widehat{a_5b_3}+a_0b_2+c_2$	$\widehat{a_4b_5}+\widehat{a_5b_4}+$ $a_0b_3+a_1b_2+c_3$	$\widehat{a_5b_5}+a_0b_4+$ $a_1b_3+a_2b_2+c_4$	$a_0b_5+a_1b_4+$ $a_2b_3+a_3b_2+c_5$	$a_1b_5+a_2b_4+$ $a_3b_3+a_4b_2+\widehat{c_0}$
t_5	b_1	$a_1b_5+a_2b_4+$ $a_3b_3+a_4b_2+$ $a_5b_1+\widehat{c_0}$	$\widehat{a_2b_5}+\widehat{a_3b_4}+$ $\widehat{a_4b_3}+\widehat{a_5b_2}+$ $a_0b_1+c_1$	$\widehat{a_3b_5}+\widehat{a_4b_4}+$ $\widehat{a_5b_3}+a_0b_2+$ $a_1b_1+c_2$	$\widehat{a_4b_5}+\widehat{a_5b_4}+$ $a_0b_3+a_1b_2+$ $a_2b_1+c_3$	$\widehat{a_5b_5}+a_0b_4+$ $a_1b_3+a_2b_2+$ $a_3b_1+c_4$	$a_0b_5+a_1b_4+$ $a_2b_3+a_3b_2+$ $a_4b_1+c_5$
t_6	b_0	$a_0b_5+a_1b_4+$ $a_2b_3+a_3b_2+$ $a_4b_1+a_5b_0+c_5$	$\widehat{a_1b_5}+\widehat{a_2b_4}+$ $\widehat{a_3b_3}+\widehat{a_4b_2}+$ $\widehat{a_5b_1}+a_0b_0+c_0$	$\widehat{a_2b_5}+\widehat{a_3b_4}+$ $\widehat{a_4b_3}+\widehat{a_5b_2}+$ $a_0b_1+a_1b_0+c_1$	$\widehat{a_3b_5}+\widehat{a_4b_4}+$ $\widehat{a_5b_3}+a_0b_2+$ $a_1b_1+a_2b_0+c_2$	$\widehat{a_4b_5}+\widehat{a_5b_4}+$ $a_0b_3+a_1b_2+$ $a_2b_1+a_3b_0+c_3$	$\widehat{a_5b_5}+a_0b_4+$ $a_1b_3+a_2b_2+$ $a_3b_1+a_4b_0+c_4$

*: Denote the register that will store specific output value w_i $(0 \leq i \leq 5)$.

final output w_i is carried out in a serial format and thus no shift-register is required. The details of this aspect of hardware structure can be seen at Section III-F.

C. Example: $W = A \cdot B$ *Mod* (x^6+1) + C

Connecting with the example given in Table I that $W = A \cdot B$ mod $(x^6+1)+C$, Table II shows the evolution in time of the contents of the k-bit registers (will store $w_0, w_1, w_2, w_3, w_4, w_5$, respectively) following the architecture of Fig. 2. We have used *Cycle* to denote the clock cycles and *Serial* to represent the serially-fed input, i.e., binary coefficient b_i, with respect to every clock cycle. Meanwhile, all the coefficients $(a_0, a_1, a_2, a_3, a_4, a_5)$ of polynomial A are fed to the structure in parallel all the time.

As shown in Table II, during the initialization process (*init.*), i.e. when $init$ = '1', all six registers are loaded with $\widehat{c_i}$ firstly (the two's complement of the coefficients c_i of polynomial C). Meanwhile, the coefficients of A are loaded in six k-bit registers, respectively, and b_5 of polynomial B is made ready to be fed to the architecture in the next cycle. In the next cycle t_1, the products $a_5b_5+\widehat{c_4}$, $a_0b_5+c_5$, $a_1b_5+\widehat{c_0}$, $a_2b_5+\widehat{c_1}$, $a_3b_5+\widehat{c_2}$ and $a_4b_5+\widehat{c_3}$ are performed and also loaded into corresponding registers, respectively. It must be noted that the first (left) adder in Fig. 2 performs the inverted two's complement of the content of register (far-left one), so the value loaded in the far-right register in cycle t_1 becomes $a_0b_5+c_5$. In cycle t_2, the bit b_4 of polynomial B is fed to the architecture and is then ANDed with the integer coefficients of A. These products are added (or subtracted for the first adder) with the values stored in the registers-(for w_i) and loaded into the following registers again. It can be observed that after the initialization process, the registers contain the exact coefficients of the results of $W = A \cdot B$ mod $(x^6+1)+C$ after six clock cycles (in t_6), matching the final values given in Table I with the addition of the c_i coefficients. For example, in t_6 the content of the specific register (for storing w_2) is $a_2b_0+a_1b_1+a_0b_2+\widehat{a_5b_3}+\widehat{a_4b_4}+\widehat{a_3b_5}+c_2 = a_2b_0+a_1b_1+a_0b_2-a_5b_3-a_4b_4-a_3b_5+c_2$.

D. Complexity Analysis of the Proposed Architecture-I

The theoretical complexity of the proposed architecture of Fig. 2 for the computation of $W = A \cdot B$ mod $(x^n+1)+C$ in $\mathcal{R}_q$ within the *Inv*BRLWE-based encryption scheme is given

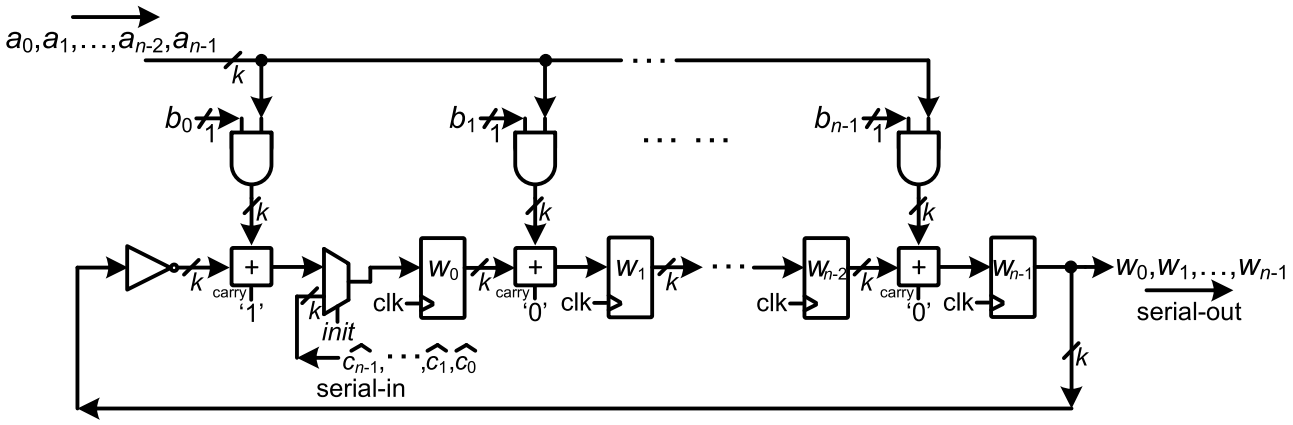


Fig. 3. Proposed architecture-II for $A \cdot B$ mod $(x^n + 1) + C$ (*Inv*BRLWE-based encryption scheme), where the values in the registers (w_{n-1}, w_0, $\cdots$, w_{n-2}) are the final accumulated results to be delivered out.

as follows. The coefficients of the integer polynomial A are stored in a n-length k-bit shift-register and the coefficients of the binary polynomial B are stored in a n-bit shift register. The coefficients of $C \in \mathcal{R}_q$ are initially loaded in w_i registers (nk 1-bit registers). Besides that, n adders (one of them acting as a subtractor) with two k-input operands are also needed for the addition operations. It is important to note that the $carry_{in}$ (*input carry*) of $(n-1)$ adders is fixed to '0' in order to perform the *addition* and one adder has $carry_{in} = '1'$ in order to perform subtraction in two's complement. These $carry_{in}$ inputs are fixed during all the computation process and no extra control signal is needed. The products of $[Ax^i]_j b_i$, with $i = 0 \ldots n-1$, are done with nk 2-input AND gates and the *one's complement* $C'_1(w_{n-1})$ is performed with k NOT gates in parallel. Furthermore, n 2:1 k-bit multiplexers (with $init$ control signal) are also needed for initialization.

With respect to the time-complexity, one can observe that the maximum combinational path-delay of this architecture is $T_{AND} + T_{ADD} + T_{MUX}$, where T_{AND}, T_{ADD} and T_{MUX} denote the delay of a 2-input AND gate, an adder and a 2:1 multiplexer, respectively. The total computation of $W = A \cdot B$ mod $(x^n + 1) + C$ requires n clock cycles (plus one extra initialization cycle to initiate the coefficients of C into the registers), not counting the input and output polynomials' loading and delivery cycles.

The detailed area-time theoretical complexities of the proposed architecture are given in Table III, where the number of 2-input AND gates (#AND), 2-input XOR gates (#XOR), inverters (#INV), adders (with two k-input operands), 2:1 multiplexers (with k-bit data inputs) and 1-bit registers or flip-flops (#FF) are included. In Table III, the needed number of clock cycles (#Clk) for the computation (once all the coefficients are stored in the corresponding registers) and the maximum combinational path-delay ($Delay$) are also given.

E. Proposed Architecture-II: Mostly Serial-In Serial-Out

1) Definition: We define that a structure is mostly serial-in serial-out when the coefficients of the major polynomials are fed in (or delivered out) with a serial format.

For resource-constrained applications that requires less area usage of the implemented PQC scheme, we can have the architecture-II as proposed in Fig. 3. This new proposal removes all the multiplexers used in architecture-I for all the coefficients' (C) initialization process except the most left one. Besides that, we have interchanged the feed-in positions of inputs A and B such that the original shift-register (k-bit size) for A in Fig. 2 is now replaced by the shift-register for B (1-bit size), which reduces the involved hardware usage. Note that the interchanging of A and B does not affect the output delivering, as shown by the example in Table II. In this new architecture, all the coefficients of polynomial C are initiated through the only multiplexer in a serial format when $init = '1'$, i.e., from $\widehat{c_{n-1}}$, ..., $\widehat{c_0}$. After n cycles, all the registers in architecture-II are initiated with the values the same as those in architecture-I through parallel initiation, namely $\widehat{c_{n-1}}$, $\widehat{c_0}$, ..., $\widehat{c_{n-2}}$ (from left to right). When the selecting signal of the multiplexer $init$ switches to '0', architecture-II works the same as that of Fig. 2 to obtain the desired results to be stored in respective registers after n clock cycles, as shown in Fig. 3 (from left to right: w_{n-1}, w_0, ..., w_{n-2}). Finally, all the results stored in these registers will be delivered out in a serial format (as shown in Fig. 3, where the output of register-w_{n-1} is attached to the outside as the only output channel for the proposed architecture-II), i.e., from w_{n-1} to w_0. Similar to Fig. 2, the output of each AND cell will be set as zero during the output delivery process that no extra values will be accumulated. Meanwhile, as the output values do not go through the inverter (NOT gates in the left side of the architecture), the correct output can be delivered in a desired format (the whole output delivery takes n cycles).

The theoretical area-time complexities of the proposed architecture-II are almost the same as those given in Subsection III-D except with only one multiplexer. Furthermore, the maximum combinational path delay in this case is still $T_{AND} + T_{ADD} + T_{MUX}$. Table III gives the complexities of the proposed architecture-II of Fig. 3.

F. Final Implementation Consideration

Due to limited number of processing bit-width (or input/output ports) in general application environments, it is usually difficult to have all the necessary coefficients directly

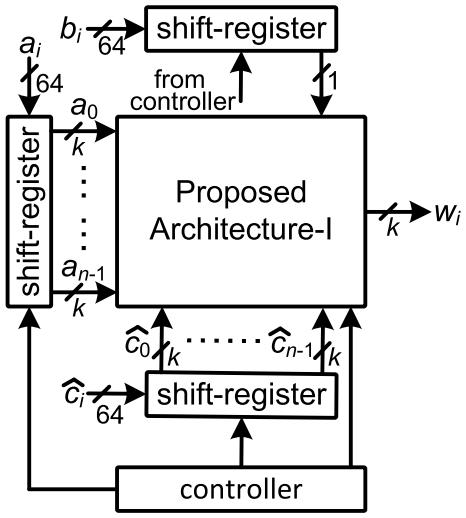


Fig. 4. Actual implementation consideration for $A \cdot B \bmod (x^n + 1) + C$ (based on architecture-I), where the input is set as 64-bit.

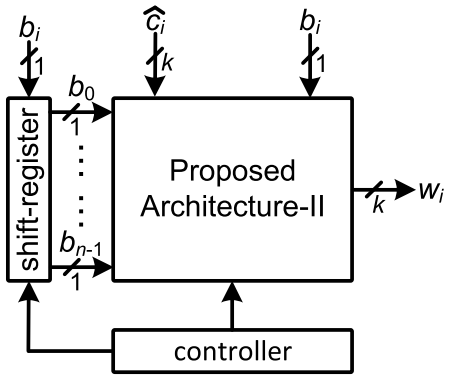


Fig. 5. Actual implementation consideration for $A \cdot B \bmod (x^n + 1) + C$ (based on architecture-II).

fed to the architecture in a parallel format (e.g., consider the fact that when $n = 256$ and $k = 8$ for the coefficients of A that needs 2,048 parallel bits). Hence, a complete top-level architecture setup for actual implementation is also presented for the computation of $W = A \cdot B \bmod (x^n + 1) + C$ in $\mathcal{R}_q$ with $q = 2^k$. Figs. 4 and 5 show the block diagram of the proposed approach, where the major component corresponds with the proposed architectures given in Fig. 2 or Fig. 3.

Conventionally, for the proposed architecture-I (mostly parallel-in serial-out), the parallel coefficients of A and C (k-bits each) are obtained through serial-in parallel-out shift-registers while the output coefficients of W are delivered out serially. The coefficients of A and C are *loaded* into the shift-registers for the computation in the proposed architecture-I core within n clock cycles, as indicated in Fig. 2. After all the output coefficients become available in the registers of the proposed architecture-I, these coefficients are then delivered out in n clock cycles. Note that the coefficients of B are fed to a serial-in serial-out shift-register for serial-delivering of corresponding b_i $(0 \leq i \leq n-1)$ to the architecture core.

Practically, however, we can speed up the *loading* stage of the proposed architecture with smaller number of clock cycles, as shown in Fig. 4. Following the practical application environment that one processing word per cycle probably can carry 64-bit, which can be loaded into an $n = 256$-length ($q = 256$ and $\log_2 q = 8$) serial-in parallel-out shift-register, every cycle loading in 64-bit, with only 32 cycles. In this case, the shift-registers are designed with 64-bit loading capacities while the deliveries are either k-bit (in parallel) or 1-bit (serial). Note this type of implementation setup can also be extended to the application environment where 32/16-bit word is commonly used.

While for the proposed architecture-II (mostly serial-in serial-out), as shown in Fig. 5, almost all the input coefficients are fed to the architecture in a serial format except the input B (which takes n clock cycles). The main computation process of the structure in Fig. 5 still requires n cycles. The output coefficients of W are finally produced out serially, which requires n clock cycles.

Note that a controller, mainly a finite-state machine (FSM), is needed to generate the control signals and to determine the *loading*, *processing* and *output delivery* stages of the proposed overall architectures for actual implementations.

1) Further Architectural Difference Consideration: From the actual implementation point of view, one can see that the proposed architecture-I is more suitable for high-performance applications where the resources are abundant, as indicated by the using of large number of shift-registers. While the proposed architecture-II is more fitting resource-relatively-limited applications since it requires less actual resource usage than the one in Fig. 4. Besides that, the architecture-II in Fig. 5 does not possess the capacity to load-in 64-bit word as the coefficients of C need fixed number (n cycles) to be loaded.

Note that for the sake of a fair comparison with the competing designs in the literature, especially the proposed architecture-I, we have also included the input loading time into latency cycles to demonstrate the efficiency of this design.

IV. Comparison With Other Architectures

In this section, the proposed two architectures for the computation of $W = A \cdot B \bmod (x^n + 1) + C$ are compared with the existing *Inv*BRLWE/BRLWE-based architectures, especially those very recent designs of [24], [27], [28], on both theoretical and implementational aspects. Note that some existing works like [26], [29] belong to compact designs (suitable for resource ultra-constrained applications), we do not include them here (for a fair comparison). The design of [25] does not have a correct structural setup on sign control and we thus do not include it in comparison to avoid unnecessary confusion. Although the work in [23] also belongs to compact designs, it includes results corresponding to a parallel architecture so we include it in the comparison.

A. Complexity Analysis and Comparison

Following the complexity estimation in Section III-*D*, we have listed the area-time complexities of the major components for the proposed architecture-I and architecture-II, as shown in Table III along with those of [23], [24], [28]. Note

TABLE III
THEORETICAL AREA-TIME COMPLEXITIES OF DIFFERENT *Inv*BRLWE-BASED PQC ARCHITECTURES

	#AND	#INV	#Adder (k-bit)	#Mux (k-bit)	#Mux (2-bit)	#FF	#Clk	Critical-Path*
[23]	–	–	$2n^{\dagger}$	$n^{\dagger}$	–	nk	n	$T_{SUB}+T_{ADD}+T_{MUX3}$
[24]	nk	$k+1$	n	$n+1$	-	nk	$n+1$	$\geq T_{AND}+T_{ADD}+T_{MUX}$
[28] ($u=1$)	$(k+1)n$	$2n$	n	1	n	nk	n	$\simeq T_{AND}+T_{ADD}+T_{MUX}$
Architecture-I	nk	k	n	n	-	nk	n	$T_{AND}+T_{ADD}+T_{MUX}$
Architecture-II	nk	k	n	1	–	nk	n	$T_{AND}+T_{ADD}+T_{MUX}$

The listed area-time complexities do not include the input/output shift-register resources in actual implementation as well as control unit. The actual implementation complexity might vary when these components are added, e.g., the actual implementation complexity of [24] is larger than [28].
†: In [23], $2n$ k-bit adders refers to n k-bit adders and n k-bit subtracters, and n Mux (k-bit) refers to n k-bit MUX with 3 inputs.
*: The actual implementation critical-path might vary due to the control unit setup, where T_{AND}, T_{ADD}, T_{SUB} and T_{MUX} refer to the delay time of an AND gate, a k-bit adder, a k-bit subtracter and a k-bit MUX, respectively (note that $k=\log_2 q=\log_2 256=8$ according to the parameter setting). T_{MUX3} refers to the delay time of a k-bit MUX with 3 inputs.

that the design of [27] has used a look-up table (LUT)-like based method to construct the BRLWE-based PQC architecture, which is different from the components used in the proposed work. We thus do not list it in Table III, but will include it in the FPGA implementation based comparison, see Table V.

As shown in Table III, one can see that the proposed architectures overall have better theoretical area-time complexities than the existing ones. As it is shown that (in [28]) the design of [28] has better performance than [24], we thus mostly focus on the comparison with [28]. It is shown that the proposed architecture-I has comparable complexity with the design of [28]. However, as mentioned in Section III-*F*, the proposed architecture-I is more suitable for high-performance applications and can easily speed-up the overall processing process in practical environment (this property does not exist in the design of [28]). While considering the design of [28] with the proposed architecture-II (these two designs have similar input/output styles), the proposed design obviously involves less area usage than the existing one (e.g., smaller number of Muxes). Although the design in [23] is a coefficient-serial architecture primarily focused on lightweight applications with side-channel countermeasures, it also presents results corresponding to a not given parallel architecture. We have deduced this parallel architecture from the coefficient-serial design (excluding countermeasures) and the theoretical complexities are given in Table III. It can be observed that the proposed architectures have better theoretical area-time complexities than the design given in [23] which also needs k-bit subtractors and k-bit Muxes with 3 inputs. Overall, one can conclude that the proposed two architectures have higher efficiency in area-time complexities than the state-of-the-art designs [24], [28].

B. FPGA-Based Implementation and Comparison

In order to further demonstrate the actual performance efficiency of the proposed architectures over the most recent similar ones given in [24], [27], and [28], FPGA-based implementation results for $(n,q)=(256,256)$ ($k=8$) and $(n,q)=(512,256)$ have been provided for all the related designs. Note that these values of (n,q) are considered secure against classic and quantum attacks [32]. It has been proven that the parameters (n,q) =(256,256) provide 84 and 73 bits of classic and quantum security levels, respectively, while the values of (n,q) =(512,256) provide 190 and 140 bits, respectively [32].

The overall experimental setup is as follows:

(i) The proposed arithmetic architectures have been coded with VHDL [1] and implemented on Xilinx and Intel FPGAs, i.e., Virtex-7 XC7V2000t, Kintex-7 XC7K325t, and Stratix-V 5SGXMABN1F45C devices, respectively, through Vivado 2019.2 and Quartus Prime 17.0 (following [27], [28]). The functions of coded designs are verified through ModelSim. Note that we have used the architectures shown in Figs. 4 and 5 for final implementation.

(ii) We have obtained the related implementation results based on parameter settings $(n,q)=(256,256)$ and $(n,q)=(512,256)$, following the existing designs [24], [27], [28], [31]. The obtained results, such as the number of Slice LUTs and Slice registers (or Adaptive Logic Modules (ALMs) on Intel device), frequency (Fmax, MHz), latency cycle, delay (critical-path$\times$latency cycles), area-delay product (ADP), power, and energy per computation (EPC) are calculated and listed in Tables IV and V, respectively. Note that the results in [27] mostly focus on the number of ALMs, Fmax, and ADP, we thus follow this style to list the related items (see Table V). Besides that, we have also listed the latency cycles, including the shift-registers' input loading, in $(\cdot)$ along with related area-time complexities for the recently released and proposed designs, as shown in Tables IV and V, respectively. Note that the designs of [24], [27] assume the input polynomials are directly fed to the structures, we thus do not calculate their latency cycles with input loading included.

From the experimental results given in Table IV for Xilinx FPGA platform, it can be observed that, for $(n,q)=(256,256)$ and $(n,q)=(512,256)$, the architecture given in Fig. 3 presents the best area-time complexities among all the reported results, e.g., it involves 5.80% and 9.67% less ADP than the very recent architecture of [28] (with similar input/output processing styles) on the Virtex-7 device, respectively. Note that the proposed architecture given in Fig. 2 is more suitable for high-performance applications (as

[1]The source code is available at: https://www.ece.villanova.edu/jxie02/lab/

TABLE IV
COMPARISON OF FPGA IMPLEMENTATION PERFORMANCE (XILINX DEVICES)

design	n	phase*	device	LUT	FF	Slice	Fmax	latency[1]	delay	ADP[2]	power	EPC[3]
[23]	256	Dec.	Spartan-6	6,728	6,813	1,874	101	262	2,594	4,861	-	-
[24][$]	256	Dec.	Virtex-7	5,153	2,151	1,701	261	257	985	1,675	921	3.529
[28] $u = 1$	256	Dec.	Virtex-7	3,600	2,568	1,146	415	256 (512)	617 (1,234)	707 (1,414)	233	0.561
Architecture-I	256	Dec.	Virtex-7	5,309	6,446	1,948	400	256 (288)	640 (720)	1,247 (1,403)	642	1.605
Architecture-II	256	Dec.	Virtex-7	2,580	2,340	907	349	256 (512)	734 (1,467)	666 (1,331)	140	0.401
[24][$]	512	Dec.	Virtex-7	10,285	4,249	3,289	263	513	1,951	6,417	1,871	7.114
[28] $u = 1$	512	Dec.	Virtex-7	7,184	5,128	2,208	399	512 (1,024)	1,283 (2,566)	2,833 (5,666)	456	1.143
Architecture-I	512	Dec.	Virtex-7	11,123	12,851	3,668	357	512 (576)	1,434 (1,613)	5,260 (5,916)	646	2.595
Architecture-II	512	Dec.	Virtex-7	5,650	4,656	1,894	379	512 (1,024)	1,351 (2,702)	2,559 (5,118)	420	1.108
[28] $u = 1$	256	Dec.	Kintex-7	3,600	2,568	1,134	394	256 (512)	650 (1,299)	737 (1,473)	237	0.602
Architecture-I	256	Dec.	Kintex-7	5,324	6,469	1,781	357	256 (288)	717 (807)	1,277 (1,437)	401	1.123
Architecture-II	256	Dec.	Kintex-7	2,836	2,340	960	373	256 (512)	686 (1,373)	659 (1,318)	193	0.517

[$]: The authors of [28] have re-implemented the design in [24] (including all input processing shift-registers), we thus use the data from [28]. Note that the authors of [28] have added additional shift-registers for practical implementation on FPGA devices as the original design assume all the input data are fed to the structure in parallel (too many I/Os). The reported LUT, FF, and slice do not included the additional shift-registers. While the extra added component takes a large portion of the I/O power, e.g.,, the I/O power is 567mW when the total dynamic power is 921mW for $n = 256$. Therefore, the power consumption of the re-implemented [24] is reported as a whole, for the benefit of potential readers' better understanding of that architecture.
*: All the related calculation for all designs are based on the decryption phase of the BRLWE-based encryption scheme (Dec.: decryption).
[1]: Latency cycles, refer to the main computation time of the designated architecture. We have also included the shift-registers' input loading in $(\cdot)$ for [28] and the proposed architectures to demonstrate the area-time efficiency of the proposed architecture-I.
Unit for Fmax: MHz. Unit for delay: ns. Unit for power (dynamic): mW. Delay=critical-path$\times$latency.
[2]: ADP=#Slice$\times$delay (Dec.) $\times 10^3$.
[3]: EPC: energy per computation=power/(Fmax $\times$ #output coefficient per cycle (Dec.)).

introduced in Section III-F), and hence is more reasonable to compare this design with the existing ones when the input loading time is included. Again, as shown in Table IV, the proposed architecture-I has 20.42% less ADP than the structure of [24] for $n = 256$ (when input loading time is not included). Meanwhile, when the input loading time is included, the proposed architecture-I has better ADP than the one of [28]. Similar situation happens to the performance results under another parameter setting or other devices. For instance, for $n = 256$ on the Kintex-7 FPGA device, the proposed architecture in Fig. 3 has 10.58% less ADP than the existing one of [28]. Besides that, the power consumption (dynamic power) of the proposed architectures are also better than the existing designs, as shown by the metrics of power and EPC in Table IV (the smaller, the better). Finally, we want to mention that the main control component for the design in [28] is a serial-in serial-out shift-register (for sign control), which may leads to a higher frequency as shown in Table IV. However, this kind of setup requires external resources to deliver $(u - 1)$ number of '1' to the sign control based shift-register. While the proposed architectures use a FSM-based controller to coordinate the overall operation, and hence the proposed structures involve more complete hardware design setups than the state-of-the-art solution in [28].

For Intel FPGA (Stratix-V)-based implementation results shown in Table V, the architectures given in Figs. 2 and 3 have similar performance as those in Table IV. For $(n, q) = (256, 256)$, Fig. 3 has much better area and timing results than [24], [27], [31], i.e., at least 71.23% less ADP than the recent one in [27]. Similarly, for $(n, q) = (512, 256)$, Fig. 3 involves 60.98% less ADP than the design of [31]. Again, we want to mention that the proposed architecture in Fig. 2 aims to be deployed for high-performance applications and hence involves more shift-registers than the other designs, as shown in Fig. 4. However, as shown in Table V, when the input loading time is included, the proposed architecture-I involves at least 34.03% and 28.83% less ADP than [31] for $n = 256$ and $n = 512$, respectively.

Overall, the proposed architecture-II (Fig. 3) has the least area-time complexities among all the reported designs for *Inv*BRLWE-based PQC. Due to its low-complexity feature, this architecture is desirable for applications in resource relatively limited environments. While the proposed architecture-I involves a novel setup on input data loading, as discussed in Section III-D, and can be deployed for high-performance applications with shortened computation cycles on input loading (and even output delivery, if the output is similarly designed).

C. Discussion

While this paper aims to deliver efficient hardware architectures for BRLWE/*Inv*BRLWE-based PQC scheme, other aspects of work such as side-channel attack and comparison with other standard Ring-LWE/LWE designs are out of the scope of this work. Nevertheless, the proposed two architectures have stable operational frequency and hence are resistant against regular timing attacks. Meanwhile, we believe the power attack based measurements developed in [23] and [38] are applicable to the proposed work here and can be one of our future research directions.

Besides that, future work also include: (i) developing more efficient arithmetic innovation to further reduce the computational complexity of the major operations involved within BRLWE/*Inv*BRLWE-based PQC scheme; (ii) conducting ultra low-complexity implementation oriented structural exploration

TABLE V
COMPARISON OF THE AREA-TIME COMPLEXITIES FOR THE PROPOSED AND COMPETING DESIGNS (INTEL STRATIX-V DEVICE)

design	ALMs	Fmax	latency[2]	delay	ADP[3]
$n = 256$ (decryption phase)					
[24][1]	5,734	369.14	257	696	3,991
[27]	4,495	321.03	258	804	3,614
[31]	4,446	379.22	257 (513)	678 (1,353)	3,014 (6,015)
A-I	4,112	298.42	256 (288)	858 (965)	3,528 (3,968)
A-II	2,315	569.8	256 (512)	449 (899)	1,039 (2,081)
$n = 512$ (decryption phase)					
[24][1]	11,470	336.36	513	1,525	17,492
[27]	9,038	317.06	514	1,621	14,651
[31]	8,864	327.98	513 (1,025)	1,564 (3,125)	13,863 (27,700)
A-I	9,769	285.39	512 (576)	1,794 (2,018)	17,526 (19,714)
A-II	4,620	437.25	512 (1,024)	1,171 (2,342)	5,410 (10,820)

A-I: Architecture-I; A-II: Architecture-II.
Unit for Fmax: MHz. Unit for delay: ns.
[1]: High-speed one (Fig.4 of [24]) and the data is obtained from the re-implemented results in [27].
[2]: refer to the main computation time of the designated architecture. We have also included the shift-registers' input loading in $(\cdot)$ for [31] and the proposed architectures to demonstrate the area-time efficiency of the proposed architecture-I.
[3]: ADP=#ALM$\times$delay$\times 10^3$.

to finalize optimal performance tradeoff between different structural setups. Meanwhile, the operand of B in Fig. 2 (or A in Fig. 3) can be decomposed into multiple groups for parallel processing to further speed-up the processing speed and one of our future work may also focus on finalizing the optimal number of processing groups.

V. CONCLUSION

In this paper, two efficient LFSR-based architectures for the arithmetic operation used in the *Inv*BRLWE-based encryption scheme, a very recent variant of BRLWE-based PQC, have been presented. The first proposed LFSR-based arithmetic architecture performs the operation of $A \cdot B + C$ with fast input loading and optimized resource usage, while the second proposed LFSR-based architecture involves even smaller area usage with a novel serial-in serial-out setup. Due to the input processing differences exist between two proposed designs, we have further developed corresponding implementation strategy for them. Overall, both theoretical analysis and FPGA-based implementation results show that the proposed architectures achieve better area-time complexities than similar schemes/structures found in the literature. In particular, it is shown that the proposed architecture-II provides the best area-time complexities among all the implemented structures, i.e., has significantly less ADP (e.g., 71.23%) than the best state-of-the-art designs on both Xilinx and Intel FPGA devices. The proposed architectures are highly efficient and can be extended for actual BRLWE/*Inv*BRLWE-based cryptoprocessor implementation in many emerging applications.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers' constructive comments to improve the quality of this article.

REFERENCES

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.
[2] J. Bernstein, J. Buchmann, and E. Dahmen, *Post-Quantum Cryptography*. Berlin, Germany: Springer, 2009.
[3] J. Hoffstein, J. Pipher, and J. Silverman, "NTRU: A ring-based public key cryptosystem," in *Proc. 3rd Int. Symp. ANTS*, 1998, pp. 267–288.
[4] R. McEliece, "A public-key cryptosystem based on algebraic coding theory," Deep Space Netw. Prog. Rep., Jet Propuls. Lab., California Inst. Technol., Pasadena, CA, USA, Tech. Rep. 42–44, 1978.
[5] S. Rohde, T. Eisenbarth, E. Dahmen, J. Buchmann, and C. Paar, "Fast hash-based signatures on constrained devices," in *Proc. CARDIS*, in Lecture Notes in Computer Science, vol. 5189, 2008, pp. 104–117.
[6] D. Jao and L. De Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *Proc. Int. Workshop Post-Quantum Cryptogr.*, 2011, pp. 19–34.
[7] J. Ding, J. E. Gower, and D. S. Schmidt, *Multivariate Public Key Cryptosystems* (Advances in Information Security), vol. 25. NewYork, NY, USA: Springer, 2006.
[8] W. Dai, W. Whyte, and Z. Zhang, "Optimizing polynomial convolution for NTRUEncrypt," *IEEE Trans. Comput.*, vol. 67, no. 11, pp. 1572–1583, Nov. 2018.
[9] J. L. Imaña and I. Luengo, "High-throughput architecture for post-quantum DME cryptosystem," *Integration*, vol. 75, pp. 114–121, Nov. 2020.
[10] A. Sarker, M. M. Kermani, and R. Azarderakhsh, "Fault detection architectures for inverted binary ring-LWE construction benchmarked on FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 4, pp. 1403–1407, Apr. 2021.
[11] D. Micciancio, "Lattice-based cryptography," in *Encyclopedia of Cryptography and Security*. Boston, MA, USA: Springer, 2011.
[12] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, p. 34, Sep. 2009.
[13] J. Buchmann, F. Göpfert, R. Player, and T. Wunderer, "On the hardness of LWE with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack," in *Proc. Int. Conf. Cryptol. Afr.*, 2016, pp. 24–43.
[14] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2010, pp. 1–23.
[15] J. Howe, C. Moore, M. O'Neill, F. Regazzoni, T. Güneysu, and K. Beeden, "Lattice-based encryption over standard lattices in hardware," in *Proc. 53rd Annu. Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.
[16] D. D. Chen *et al.*, "High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 1, pp. 157–166, Jan. 2015.
[17] Y. Zhang, C. Wang, D. E. S. Kundi, A. Khalid, M. O'Neill, and W. Liu, "An efficient and parallel R-LWE cryptoprocessor," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 5, pp. 886–890, May 2020.
[18] W. Liu, S. Fan, A. Khalid, C. Rafferty, and M. O'Neill, "Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 10, pp. 2459–2463, Oct. 2019.
[19] D. Liu, C. Zhang, H. Lin, Y. Chen, and M. Zhang, "A resource-efficient and side-channel secure hardware implementation of ring-LWE cryptographic processor," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 4, pp. 1474–1483, Apr. 2018.
[20] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Instruction-set accelerated implementation of CRYSTALS-kyber," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4648–4659, Nov. 2021.
[21] C. P. Rentería-Mejía and J. Velasco-Medina, "High-throughput ring-LWE cryptoprocessors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2332–2345, Aug. 2017.
[22] J. Buchmann, F. Göpfert, T. Güneysu, T. Oder, and T. Pöppelmann, "High-performance and lightweight lattice-based public-key encryption," in *Proc. ACM Int. Workshop IoT Privacy, Trust, Secur.*, May 2016, pp. 1–8.
[23] A. Aysu, M. Orshansky, and M. Tiwari, "Binary ring-LWE hardware with power side-channel countermeasures," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1259–1264.

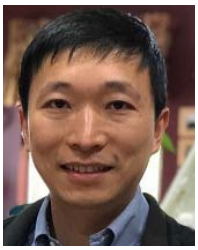
[24] S. Ebrahimi, S. Bayat-Sarmadi, and H. Mosanaei-Boorani, "Post-quantum cryptoprocessors optimized for edge and resource-constrained devices in IoT," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5500–5507, Jun. 2019.
[25] J. Xie, K. Basu, K. Gaj, and U. Guin, "Special session: The recent advance in hardware implementation of post-quantum cryptography," in *Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2020, pp. 1–10.
[26] P. He, U. Guin, and J. Xie, "Novel low-complexity polynomial multiplication over hybrid fields for efficient implementation of binary ring-LWE post-quantum cryptography," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 2, pp. 383–394, Jun. 2021.
[27] J. Xie, P. He, and W. Wen, "Efficient implementation of finite field arithmetic for binary ring-LWE post-quantum cryptography through a novel lookup-table-like method," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 1279–1284.
[28] J. Xie, P. He, X. M. Wang, and J. L. Imana, "Efficient hardware implementation of finite field arithmetic $AB + C$ for binary ring-LWE based post-quantum cryptography," *IEEE Trans. Emerg. Topics Comput.*, early access, Jun. 23, 2021, doi: 10.1109/TETC.2021.3091982.
[29] K. Shahbazi and S.-B. Ko, "Area and power efficient post-quantum cryptosystem for IoT resource-constrained devices," *Microprocessors Microsyst.*, vol. 84, Jul. 2021, Art. no. 104280.
[30] S. Ebrahimi and S. Bayat-Sarmadi, "Lightweight and fault-resilient implementations of binary ring-LWE for IoT devices," *IEEE Internet Things J.*, vol. 66, no. 1, pp. 203–204, Aug. 2020.
[31] B. J. Lucas *et al.*, "Lightweight hardware implementation of binary ring-LWE PQC accelerator," *IEEE Comput. Archit. Lett.*, vol. 21, no. 1, pp. 17–20, Jan./Jun. 2022.
[32] F. Göpfert, C. Van Vredendaal, and T. Wunderer, "A hybrid lattice basis reduction and quantum search attack on LWE," in *Proc. Int. Workshop Post Quantum Cryptogr. (PQCrypto)*, Utrecht, The Netherlands, Jun. 2017, pp. 184–202.
[33] A. H. Namin, H. Wu, and M. Ahmadi, "A new finite-field multiplier using redundant representation," in *IEEE Trans. Comput.*, vol. 57, no. 5, pp. 716–720, May 2008.
[34] F. Rodríguez-Henríquez, N. Saquib, A. Díaz-Pérez, and Ç. K. Koç, *Cryptographic Algorithms on Reconfigurable Hardware*. New York, NY, USA: Springer, 2006.
[35] J. L. Imaña, "LFSR-based bit-serial $GF(2^m)$ multipliers using irreducible trinomials," *IEEE Trans. Comput.*, vol. 70, no. 1, pp. 156–162, Jan. 2021.
[36] B. Liu and H. Wu, "Efficient architecture and implementation for NTRUEncrypt system," in *Proc. IEEE 58th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2015, pp. 1–4.
[37] H. Wu and X. Gao, "Efficient multiplier and FPGA implementation for NTRU prime," in *Proc. IEEE Can. Conf. Electr. Comput. Eng. (CCECE)*, Sep. 2021, pp. 1–5.
[38] T. Schneider, A. Moradi, and T. Güneysu, "Part I—Towards combined hardware countermeasures against side-channel and fault-injection attacks," in *Proc. Annu. Cryptol. Conf.*, Oct. 2016, pp. 302–332.

José L. Imaña received the M.Sc. and Ph.D. degrees in physics from Complutense University of Madrid, Madrid, Spain, in 1989 and 2003, respectively. He was an Electronic Design Engineer with the Madrid Institute of Technology, Spain. He is currently with the Department of Computer Architecture and Automation, Complutense University of Madrid, where he was promoted to an Associate Professor with tenure in 2006. He has been the Promoter and a Co-Founder of the International Workshop on the Arithmetic of Finite Fields (WAIFI). His research interests include algorithms and VLSI architectures for computations in finite fields, computer arithmetic, cryptographic hardware, and post-quantum cryptography.

Pengzhou He received the B.S. degree in intelligence science and technology from the University of Science and Technology Beijing in 2019. He is currently pursuing the Ph.D. degree in computer engineering with Villanova University. His research interests include post-quantum cryptography, hardware security, high-performance IoT devices, and the application of machine learning in security systems.

Tianyou Bao received the M.S. degree in computer science from George Washington University in 2021. He is currently pursuing the Ph.D. degree in computer engineering with Villanova University. His research interests include post-quantum cryptography, hardware security, and the application of machine learning in security systems.

Yazheng Tu received the M.S. degree in computer science from Washington University in St. Louis in 2020. He is currently pursuing the Ph.D. degree in computer engineering with Villanova University. His research interests include lightweight post-quantum cryptography, hardware security, and AI in security systems.



Jiafeng (Harvest) Xie (Senior Member, IEEE) received the M.E. degree from Central South University in 2010 and the Ph.D. degree from the University of Pittsburgh in 2014.

He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA. His research interests include cryptographic engineering, hardware security, post-quantum cryptography, and VLSI implementation of neural network systems.

Dr. Xie has served as a Technical Committee Member for many reputed conferences, such as HOST, ICCAD, and DAC. He received the IEEE ACCESS Outstanding Associate Editor for the year of 2019 and the Best Paper Award from IEEE HOST'19. He is also currently serving as an Associate Editor for *Microelectronics Journal* and IEEE ACCESS. He was serving as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS.