Open-Source FPGA Implementation of Post-Quantum Cryptographic Hardware Primitives

Rashmi Agrawal, Lake Bu, Alan Ehret, and Michel Kinsy
Adaptive and Secure Computing Systems Laboratory
Department of Electrical and Computer Engineering, Boston University
Email: {rashmi23, bulake, ehretaj, mkinsy}@bu.edu

***Abstract*—The development and implementation of post-quantum cryptosystems have become pressing issues in the design of secure computing systems, as general quantum computers have become more feasible in recent years. In this paper, we introduce a set of FPGA-based post-quantum cryptographic primitives (PQCPs) consisting of four frequently used security components, i.e., public key cryptosystem (PKC), key exchange (KEX), oblivious transfer (OT), and zero-knowledge proof (ZKP). The three main contributions of this work are: (1) FPGA-tailored implementation of the hardware primitives with novel algorithmic proposals of the OT and ZKP; (2) algorithmic optimizations to reduce area and latency costs without compromising security; and (3) open-sourcing the synthesizable and fully verifiable code for the community at large. The RTL code base is fully parameterizable with an efficient, n-point Number-Theoretic Transform (NTT) module for fast polynomial multiplications. These primitives will aid researchers and designers in constructing quantum-proof secure computing systems to prepare for the post-quantum era. Implementation results, on an Zynq-7000 FPGA, show various design trade-offs and correlations between system parameters and the associated hardware cost and latency. The source code for this project is available on the ASCS Lab website at the following URL: http://ascslab.org/research/pqcp/index.html.**

***Index Terms*—Post-quantum cryptography, FPGA-based prototyping, public-key cryptosystem, key exchange, oblivious transfer, zero-knowledge proof.**

I. Introduction

The recent development trend [1] [2] [3] [4] [5] in the field of quantum computers has confirmed that it is only a matter of time before these computer systems become functional and readily available. Quantum computers hold the promise of a significant computational power increase. These computer systems will be able to compute efficiently solutions for many computational problems that are NP-hard on conventional machines. While this development presents many compute opportunities, it also deepens our current cyber-security crisis by making many of the classical cryptosystems non-secure or critically weakened. For instance, with quantum algorithms capable of efficiently solving the integer factorization and discrete logarithm problems, RSA, ECC and ElGamal will all need to be re-examined or even replaced, since these computational problems form the core of their security. In fact, research efforts to develop a new class of post-quantum algorithms and cryptosystems are now underway.

In early 2017, the National Institute of Standards and Technology (NIST) launched a campaign [6] to standardize the post-quantum cryptography. In the first round of submissions, 69 candidate algorithms were put forth, with 27 advancing to the second round. The most commonly used algorithmic approach across these submissions is the Ring-Learning with Errors (Ring-LWE) [7] method. In fact, Ring-LWE is used in 12 out of 27 second round candidate submissions. Ring-LWE cryptosystems have a number of key advantages: (1) its lattice-based security reduction – modified shortest vector problems – remains NP-hard even on quantum computers, (2) it has a much smaller key size compared with other techniques of comparable security guarantees, (3) it supports homomorphic encryption, and (4) it could lend itself to efficient hardware implementations.

In this work, we leverage these capabilities and develop efficient hardware implementations. Although there are myriad works exploring different implementations of the Ring-LWE algorithm in software, hardware level design space exploration efforts have been very tentitave. Moreover, out of the existing hardware implementations, very few focus on scalability and efficiency. One technical reason for this is that large polynomial operations over finite rings – which form the core computational kernel of Ring-LWE algorithms – remain a key challenge for many hardware designers. To address this issue, we introduce a set of highly-optimized, parameterizable hardware modules to serve as primitives for faster design space exploration of post-quantum cryptosystems, especially the systems using Ring-LWE algorithms.

The post-quantum hardware primitive set consists of four frequently used security components: the public key cryptosystem (PKC) [8] [9], key exchange (KEX) [10] [11] [12], oblivious transfer (OT), and zero-knowledge proof (ZKP). The PKC and KEX form the basis of most modern cryptographic systems. The OT is used in many privacy-preserving applications, e.g., DNA database query and private machine learning [13]. Similarly, ZKP is used in a number of applications, such as a potential candidate for the next generation of blockchain [14] algorithms. The major contributions of our work are:

1) **Implementation:** FPGA-tailored implementation of the primitives with novel algorithmic proposals of the OT and ZKP primitives.
2) **Optimization:** Judicious algorithmic optimizations to reduce the area and power cost without compromising security.

3) **Open Source:** Release of the synthesizable and fully verifiable Verilog code for the community at large. The code base provides unique advantages including:
   a) **Parameterization:** A parameterizable design to generate variably-sized primitives to enable their deployment in small devices (e.g., IoTs) as well as large platforms (e.g., homomorphic encryption engines).
   b) **Fast Polynomial Multiplier:** An efficient FPGA implementation of a n-point Numeric-Theoretic Transfer (NTT) based high speed polynomial multiplier to be used in all primitives.

These primitives will serve as the fundamental building blocks to aid hardware designers in constructing quantum-proof secure systems to prepare for the post-quantum era.

II. The Algorithms of the Post-quantum Primitives

This section briefly introduces the four post-quantum primitives through their corresponding algorithms. It is worth noting that the algorithms for PKC [7] and KEX [11] are from widely accepted works, and the algorithms for OT and ZKP are part of our novel proposal.

First, all the algorithms are based on the same setup of a polynomial ring as follows:

Definition 2.1: Let the ring R_q be $R_q = R/\langle q\rangle = \mathbb{Z}_q[x]/\langle f(x)\rangle$, where $f(x) = x^n + 1$ is an irreducible polynomial with n a power of 2, and $q \equiv 1 \mod 2n$ is a large prime number. Thus R_q is a ring of integer polynomials modulo both $f(x)$ and q, and it has q^n elements. Let $\mathcal{X}$ be a discrete Gaussian distribution of small errors/noise centered around zero with standard deviation αp, where $\alpha < \sqrt{\log n/n}$.

A. The Public-Key Cryptosystem (PKC)

The Public-Key cryptosystem is an asymmetric encryption scheme involving a non-identical key pair, i.e., a public key and a private key. The public key is used for encryption and the private key is used for decryption. The main advantage of PKC is that anyone with the public key can encrypt messages, but only the key distributor is able to decrypt the cipher. Digital signature is one of the most prevalent application of this cryptosystem. The PKC algorithm works as follows:

B. The Key Exchange (KEX)

The PKC can be expensive for encrypting long plaintexts, due to asymmetric encryption's complexity. Hence, a key exchange mechanism that facilitates the key establishments for symmetric cryptosystems, is a proper choice. The New Hope algorithm [15], a Ring-LWE based key exchange mechanism, is a widely adopted post-quantum KEX algorithm that ensures uniform distribution of the agreed key. Since it can be constructed using the same sub-modules as the PKC, we skip its algorithm introduction for brevity. The detailed algorithm can be found in [10].

C. The Oblivious Transfer (OT) Primitive

We propose a simple OT algorithm on the foundation of the PKC primitive. The OT mechanism enables a receiver to choose and receive a certain piece of information out of many pieces from the sender, while remaining oblivious to the other pieces. The sender is also oblivious to the exact piece selected. The OT is a widely used protocol in privacy-preserving computations between two or more parties.

We denote KeyGen() as the *Key Generation* module, $\text{Enc}_{pk}()$ as the *Encryption* function, and $\text{Dec}_{sk}()$ as the *Decryption* module as seen in Algorithm 1. The proposed OT algorithm over ring R_q is as follows:

Algorithm 1: R-LWE Public Key Cryptosystem

Setup: Let $t = \lfloor \frac{q}{2} \rfloor$, $a, b \in R_q$ and $s, e, r_0, r_1, r_2 \leftarrow \mathcal{X}$, then the public key encryption protocol between Alice and Bob:

1 **Key Generation:** Alice picks a random $a \in R_q$ and samples $s, e \leftarrow \mathcal{X}$ to generate the public key $pk = \{a, b\}$ and the private key $sk = \{s\}$ by:

$$b = a \cdot s + e \tag{1}$$

where $\cdot$ is polynomial multiplication over the ring. Alice sends $\{a, b\}$ to Bob and keeps s to herself.

2 **Encryption:** Bob samples $r_0, r_1, r_2 \leftarrow \mathcal{X}$. He then converts his message into a binary vector (plaintext) m of length n, and generates the cipher $\{c_0, c_1\}$ as:

$$\begin{cases} c_0 = b \cdot r_0 + r_2 + tm, \\ c_1 = a \cdot r_0 + r_1. \end{cases} \tag{2}$$

3 **Decryption:** Alice decrypts the cipher by:

$$m = \lceil (c_0 - c_1 \cdot s)/t \rfloor, \tag{3}$$

where $\lceil \quad \rfloor$ stands for taking the nearest binary integer.

Algorithm 2: Oblivious Transfer Based on R-LWE Public Key Encryption

Setup: Let KeyGen() be the Key Generation function of the sender Alice, Enc() the encryption function of the receiver Bob, and Dec() the decryption function of Alice as in Algorithm 1. Alice has N n-bit binary messages $\{m_1, \cdots, m_N\}$ that Bob can choose from, and N n-byte random vectors $\{r_1, \cdots, r_N\}$ where $r_i \in R_q$. Then the oblivious transfer between Alice the sender and Bob the receiver is as follows:

1 Alice sends $\{r_1, \cdots, r_N\}$ to Bob. Bob chooses the c^{th} vector r_c in order to acquire m_c. Then Bob generates a random binary vector $K \in R_2^n$ and computes v to send to Alice:

$$v = r_c + \text{Enc}_{pk}(K), \tag{4}$$

where r_c is added to both ciphertexts $\{c_0, c_1\}$ (ref. [Eq.2])

2 For all $i \in \{1, 2, \cdots, N\}$, Alice computes the set $\{m_i'\}$ and sends it back to Bob:

$$m_i' = \text{Dec}_{sk}(v - r_i) \oplus m_i \tag{5}$$

where r_i is subtracted from both ciphertexts $\{c_0, c_1\}$ (ref. [Eq.2]) and $\oplus$ is bitwise XOR.

3 Bob computes his desired m_c while remaining oblivious to other m_i, where $i \neq c$:

$$m_c = m_c' \oplus K. \tag{6}$$

D. The Zero-Knowledge Proof (ZKP) Primitive

We propose a simple two-round ZKP algorithm in this subsection. The ZKP enables an entity to prove to a verifier that it knows a secret value s, without revealing any information (including the value of s) apart from the fact that it knows the value. Next generation blockchain applications leverage ZKP protocol to maintain security. Like the OT primitive, the ZKP primitive can be designed using the building blocks of the

PKC algorithm. The proposed ZKP over ring R_q is given in Algorithm 3.

Algorithm 3: Zero-Knowledge Proof Based on R-LWE

Setup: Let $t = \lfloor \frac{q}{2} \rfloor$, $a, b, s \in R_q$ and $e, r, e', u \leftarrow \mathcal{X}$.
Suppose Alice has a secret s and needs to prove her ownership of it to Bob. It is notable that unlike the PKC scheme where $s \leftarrow \mathcal{X}$, in this ZKP scheme s can be any arbitrary value as $s \in R_q$.

1 Alice picks a random $a \in R_q$ and samples $e, e', r \leftarrow \mathcal{X}$. Alice also selects an arbitrary binary vector m to compute:

$$\begin{cases} b = a \cdot s + e, \\ c = a \cdot r + mt + e' \end{cases} \tag{7}$$

where $\cdot$ is polynomial multiplication over the ring. Alice sends $\{a, b, c, m\}$ to Bob without revealing s. Bob samples $u \leftarrow \mathcal{X}$, and interactively sends it to Alice.

2 Alice responds with x to Bob:

$$x = r + s \cdot u. \tag{8}$$

3 Bob verifies if:

$$\lceil (c - a \cdot x + b \cdot u)/t \rfloor \stackrel{?}{=} m, \tag{9}$$

where $\lceil \quad \rfloor$ stands for taking the nearest binary integer. If the equality of [Eq. 9] stands, then Alice has successfully proven her ownership of s to Bob.

III. FPGA-Based Implementation

The development of a hardware module for the Ring-LWE based PKC remains complicated for designers due to the complexity and quantity of arithmetic operations involved. The challenge is thus to find an architecture that minimizes hardware costs while maximizing performance, accuracy, and parameterization. In this section, we introduce the design of one such architecture for the Ring-LWE based PKC. The design of various sub-modules will be elaborated next, along with the motivations behind the design choices for each of these sub-modules. While the bulk of the description here focuses on the PKC, the core modules are shared among all of the PQCP hardware primitives described earlier. Finally, we discuss various design trade-offs, parameterization, and the opportunities for optimization.

A. Main Algorithm

Our FPGA implementation of the PKC is based on the original Ring-LWE PKC algorithm proposed by Lyubashevsky et al. [7] as introduced in Algorithm 1.

B. Overall Architecture of The PKC

The architecture consists of three core building blocks. Each building block represents an equation in the main algorithm. Figure 1 shows the overall system architecture of the PKC using the commonly shared hardware modules.

The KeyGen module computes Eq. (1), which breaks down to vector multiplication and vector addition. The Encryption module computes Eq. (2) and consists of vector multiplication, vector addition and scalar multiplication. Similarly, the Decryption module computes Eq. (3) and consists of vector multiplication, vector subtraction and scalar division to the nearest binary integer. Hence, building the Ring-LWE based PKC comes down to constructing these basic vector operation modules.

Each n-digit vector $a = [a_0, a_1, ..., a_{n-1}]$ can also be written as: $a = a_0 + a_1 \cdot \alpha^1 + \cdots + a_{n-1} \cdot \alpha^{n-1}$, where α is the primitive root of the ring. Therefore, the operations between two vectors in a ring can also be treated as operations between two polynomials whose coefficients are the elements of the vectors.

C. Sub-modules and Design Choices

As discussed above, the basic operations of the algorithms are polynomial addition, polynomial subtraction, scalar multiplication, scalar division to the nearest binary integer, and polynomial multiplication. Most of the operations are component-wise, or can be reduced to conditional assignments. The polynomial multiplication operation has the highest hardware implementation complexity. An efficient multiplication module will substantially improve the hardware implementation efficiency of the entire hardware crypto-primitive suite.

It is also notable that all of the sub-modules involve the modulo operation, which is not cheap in hardware (a few hundred LUTs). Thus, in the following implementations, we will aim to use as few modulo operations as possible.

1) Polynomial Addition and Subtraction: Polynomial addition and subtraction are the two most frequently used modules. Both can be implemented as component-wise operations on two polynomials' coefficients, with each result being wrapped within mod q. We implement a hardware circuit for polynomial addition and polynomial subtraction as shown in Figure 2. Our implementation uses a conditional assignment to avoid using the expensive modulo operation, saving a great number of LUTs.

2) Scalar Multiplication: We again leveraged conditional assignment to implement the scalar multiplication hardware circuit shown in Figure 2. This is possible because m, the plaintext message, is an n-bit vector. Thus, computing tm is essentially choosing t or 0 according to each bit of m. Hardware cost is saved here as well by using muxes rather than performing actual multiplication operations.

3) Scalar Division to the Nearest Binary Integer: We implement the scalar division hardware circuit shown in Figure 2, without using any hardware division circuit.

We leverage the fact that the dividend and divisor are both within q and the quotient is rounded to the nearest binary integer. From Eq. (3) in the main algorithm, we denote $u = (c_0 - s \cdot c_1)$ and we know that $t = \lfloor \frac{q}{2} \rfloor$; hence, what we need to compute is $m = \lceil \frac{u}{t} \rfloor$. The nearest binary integer equivalent of $\lceil \frac{u}{t} \rfloor$ can be computed as $(\text{Absolute}(u - t) < \frac{t}{2})\ ?\ 1 : 0$. This holds true because $\text{Absolute}(u - t)$ measures the distance between u and t. If this distance is larger than half of t, it means u and t are far from each other, so the nearest integer of the quotient $\frac{u}{t}$ must be 0. But if the distance is less than half of t, then the nearest integer of the quotient $\frac{u}{t}$ must be 1.

4) Number-Theoretic Transform: A Number-Theoretic Transform is a generalization of the Fast Fourier Transform

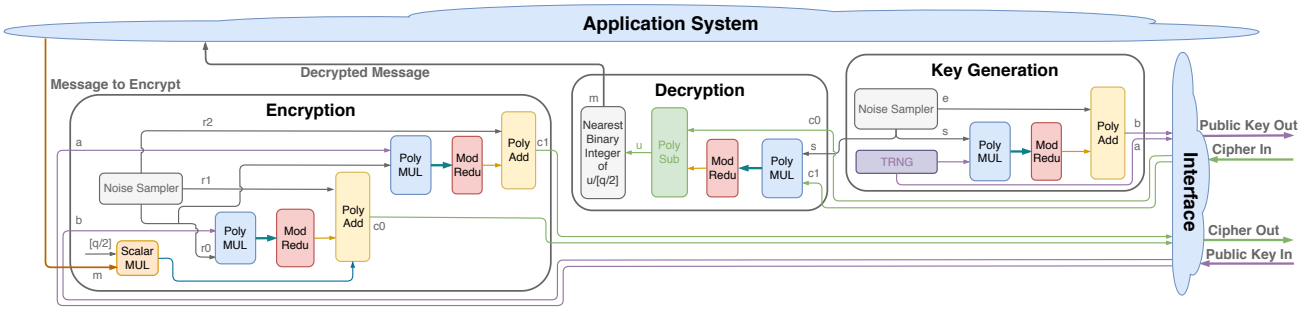

Fig. 1. The three core building blocks for the PKC algorithm: *Key Generation (KeyGen)*, *Encryption (Enc)*, and *Decryption (Dec)*.

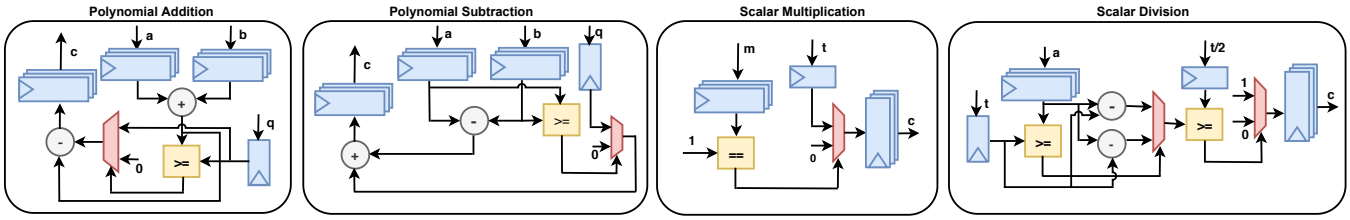

Fig. 2. The four basic operations of the PKC algorithm: *Polynomial Addition*, *Polynomial Subtraction*, *Scalar Multiplication*, and *Scalar Division to the Nearest Binary Integer*. All operations to be performed over R_q.

(FFT) over a finite ring $R_q = R/\langle q \rangle = \mathbb{Z}_q[x]/\langle f(x) \rangle$ instead of the complex number field $\mathbb{C}$. The NTT equation is given by:

$$X_i = \sum_{k=0}^{n-1} x_k \cdot \omega^{ik} \tag{10}$$

where ω is the n^{th} root of unity in the corresponding field of the polynomial. For a ring R_q, where q is a prime number, the n^{th} root of unity ω satisfies two conditions:

1) $\omega^n = 1 \bmod q$,
2) The period of ω^i for $i \in \{0, 1, 2, \cdots n-1\}$ is exactly n.

Computing ω is not trivial. One fast approach is:

1) Find the primitive root of q, which must satisfy:
   - $\alpha^{q-1} = 1 \bmod q$
   - The period of α^i for $i \in \{0, 1, 2, \cdots q-1\}$ is exactly $q-1$.
2) Since $\omega^n \equiv \alpha^{q-1} \bmod q$, we have:
$$\omega = \alpha^{(q-1)/n} \bmod q$$
3) Verify this ω according to its two conditions above.

For Inverse NTT (INTT), the computations can be performed using the NTT equation after replacing ω with ω^{-1}. ω^{-1} can be computed as $\omega^{-1} = \omega^{n-1} \bmod q$. INTT computation also requires computing the inverse of n, which can be computed as $n^{-1} \cdot n = 1 \bmod q$. Next, we will explore how NTT can be used for performing fast polynomial multiplication.

5) Polynomial Multiplication using NTT: Polynomial multiplication has the highest implementation complexity; hence, the polynomial multiplication module will govern the efficiency of the entire system. So, it is critical to design an efficient polynomial multiplication module.

One of the most common implementations of the polynomial multiplier is using convolution. However, the convolution method has a complexity of $O(n^2)$ multiplications and, therefore, is not an efficient way to implement it in hardware. A better approach is based on negative wrapped convolution combined with butterfly number-theoretic transform (NTT). This approach takes $O(n \log_2 n)$ multiplications. The NTT-based multiplication algorithm 4 proposed by Chen et al. [16] is as follows:

Algorithm 4: NTT-based Polynomial Multiplier

Setup: Let $a = \{a_0, \cdots, a_{n-1}\}$ and $b = \{b_0, \cdots, b_{n-1}\}$ $\in \mathbb{Z}_q[x]/\langle f(x) \rangle$ be two polynomials of length n (with n coefficients), where $f(x) = x^n + 1$ is an irreducible polynomial with n a power of 2, and $q \equiv 1 \bmod 2n$ is a large prime number). Let ω be the n-th root of unity, ω^{-1} the inverse of ω, $\phi^2 = \omega \bmod q$, and ϕ^{-1} the inverse of ϕ.

Precompute: $\{w^i, w^{-i}, \phi^i, \phi^{-i}\}$ for all $i \in \{0, 1, \cdots, n-1\}$
/ negative wrap convolution of a and b */*
for *i=0 to n-1* **do**
  $\bar{a}_i \leftarrow a_i \phi^i$
  $\bar{b}_i \leftarrow b_i \phi^i$
end
/ number-theoretic transforming a and b */*
$\bar{A} \leftarrow NTT^n_\omega(\bar{a})$
$\bar{B} \leftarrow NTT^n_\omega(\bar{b})$
/ component-wise multiplying A and B */*
$\bar{C} = \bar{A} \cdot \bar{B}$
$\bar{c} \leftarrow iNTT^n_\omega(\bar{C})$
for *i=0 to n-1* **do**
  $c_i \leftarrow \bar{c}_i \phi^{-i}$
end
Return c

One of the key contributions of our work is a parameterized and optimized implementation of algorithm 4. Though hardware implementations of NTT algorithms exist, they are very expensive in terms of hardware cost, power and latency. This is because prior implementations perform a great number of multiplications and divisions in computing the indices of the points and the corresponding w^i. This may not be an issue for software implementations. However, for hardware implemen-

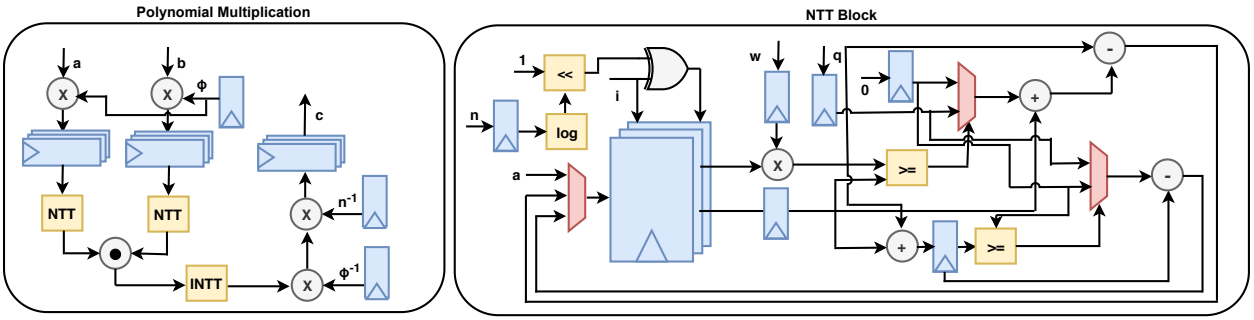

Fig. 3. *Polynomial Multiplication, Operation within NTT Block*

tations, those multiplications and divisions will consume large numbers of LUTs.

In our implementation of algorithm 5, we use only shift and xor to compute the indices of the points and the corresponding w^i. Shift and xor are cheap to implement in hardware and these simple operations replace the large multiplication and division circuits. The schematic diagram in Fig. 3 shows a high level circuit for polynomial multiplication and the operations within the NTT block. This is the most area efficient hardware implementation of an NTT algorithm we have yet seen. It should be noted that since this the algorithm is tailored for FPGA implementation and uses many bit-level operations, one may find it difficult to perform a one-on-one mapping to an optimized software implementation.

Algorithm 5: Number-Theoretic Transform Fully Optimized for Hardware Implementation

Let $a = \{a_0, \cdots, a_{n-1}\} \in \mathbb{Z}_q[x]/\langle f(x)\rangle$ where n is a power of 2, and $q \equiv 1 \bmod 2n$ is a prime number. Let ω be the n-th root of unity for q. Let ω^{-1} be the inverse of ω. Precompute ω^i and ω^{-i} for $i \in \{0, 1, \cdots, n/2-1\}$, and store them in the array element $\omega[i]$ and $i\omega[i]$ respectively.

Let a be swapped to A so that $A[j] = a[j_{rev}]$, where j_{rev} is a binary vector bit-reversed from j. Let $i, Stage$ both be initialized to 0.

Index Computation:

/ calculate the corresponding point's index i_{corr} to the i^{th} point */*

assign $i_{corr} = i$ XOR $(1 << Stage)$;

/ calculate the twiddle factor k for both i_{corr} and i */*

assign $k_0 = (Stage == 0)?0 : (i << (\log n - Stage))$;

assign $k = k_0[\log q - 1 : 1]$;

Shared Variable Computation:

assign $v = A[i_{corr}] * \omega[k] \mod q$;

NTT Function:

if $Stage < \log n$ **then**
  if $i < n$ **then**
    $i = i + 1$;
    if $i == n - 1$ **then**
      $Stage = Stage + 1$;
    end
    if $i[Stage] == 0$ **then**
      $A[i] = A[i] + v - (A[i] + v \geq q?q : 0)$;
      $A[i_{corr}] = A[i] - v + (A[i] \geq v?0 : q)$;
    end
  end
end
else
  Return A as the transformed polynomial of a.
end

D. Gaussian Noise Sampler and True Random Number Generator (TRNG)

In all of the algorithms mentioned in Section II, the source for generating small random noise vectors is a Gaussian Noise Sampler. The importance of generating noise vectors within Gaussian distribution range is described in detail by Regen [17]. Also, a TRNG is required in the KeyGen module to generate the public key a. Hence, the quality of a will be determined by the quality of TRNG. Therefore, designing and implementing an efficient Gaussian Noise Sampler and a high entropy TRNG are critical. Implementations of the Gaussian Noise Sampler and the TRNG are beyond the scope of this paper, and one can refer to Karmakar et al. [18] and Sunar et al. [19] for further details.

E. Parameterization and Open Source

A realistic implementation of the Ring-LWE based PKC requires $q > 10,000$, with each polynomial having $128 \sim 1024$ coefficients. This ensures that the algorithm has a security level of at least 112 bits, as per the security standards prescribed by NIST. Thus, parameterizing the implementation assists in easy transition to large values for q and n to achieve the appropriate long term security. Moreover, a parameterizable design also facilitates generating variably-sized primitives to enable their deployment in small devices, such as battery powered sensors or other IoT devices, as well as large computing platforms, such as homomorphic encryption engines.

As a contribution to the research community, we will open source this parameterizable Verilog code base. Additionally, we will release the software code for generating ω, n-th root of unity for q and ϕ, the negative wrapped convolution variable. This will provide researchers with a good baseline for quickly designing their own secure architecture schemes.

IV. Evaluation of Cost and Performance

We first present the correlation between the system parameters n, q and costs (latency and hardware) for all the three building blocks of PKC system. These equations provide an estimation of the performance and cost as soon as $\{n, q\}$ are selected. Therefore, it helps designers to plan ahead regarding hardware resources and latency expectations for their own systems. As shown in Table I, the latency (in clock cycles) can be precisely computed as a function of n, and the hardware cost is proportional to $n \log_2 n \log_2 q$. This estimation is backed

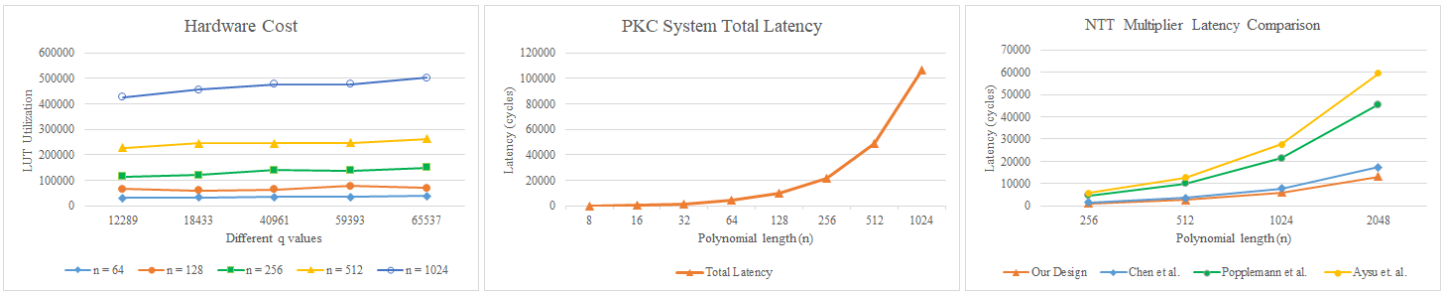

Fig. 4. (a) Hardware Cost with different q and n for PKC System, (b) Total Latency (cycles) for PKC System, and (c) Latency comparison of NTT Multiplier for our design, the design of Chen et al. [16], Pöppelmann et al. [20], and Aysu et al. [21].

by the experimental data listed in the tables and shown in the figures.

Our PKC is implemented on the Zynq-7000 FPGA. The synthesis is carried out using Xilinx Vivado 2018.2 design suite. Table II and III present the implementation results, i.e., the hardware cost for the entire PKC system. The results indicate that the hardware cost depends on the length (n) of the polynomial and size (bit-width) of the chosen prime number (q). Synthesis results are generated using variable polynomial length, n while the prime number (q) used is 12,289. This value of q is widely accepted as one of the safe to use values in Ring-LWE based cryptosystems. We also present the hardware cost trend for different combinations of n and q in Figure 4(a).

TABLE I
CORRELATION BETWEEN $\{q, n\}$ AND {LATENCY, AREA}

Operation	Latency
KeyGen	$3n + \frac{3n}{2}\log_2 n$
Enc	$7n + 2n\log_2 n$
Dec	$4n + n\log_2 n$
Resource	**Cost**
LUTs	$O(n \log_2 n \log_2 q)$
Registers	$O(n \log_2 n \log_2 q)$

TABLE II
HARDWARE COST(LUTS ONLY IMPLEMENTATION) WITH DIFFERENT n AND q=12,289

Length n	LUTs	Registers	DSP
128	66251	16805	26
256	11490	33138	26
512	227458	65643	26
1024	426402	130540	26

TABLE III
HARDWARE COST(BRAM IMPLEMENTATION) WITH DIFFERENT n AND q=12,289

Length n	LUTs	Registers	DSP	BRAM
128	7376	221	26	3.5
256	9152	396	26	3.5
512	11504	674	26	3.5
1024	15717	1255	26	3.5

In Table IV and Figure 4(b), we present the latency of the PKC system with varying polynomial length n. We can see that the latency in clock cycles is determined by n, independent of q. However, the maximum frequency will be reduced as q increases, due to a larger combinational logic in the modulo operation.

TABLE IV
LATENCY (CYCLES) FOR KEYGEN, ENC AND DEC MODULES IN PKC

Length n	KeyGen	Enc	Dec
8	96	152	80
16	240	368	192
32	576	864	448
64	1344	1984	1024
128	3072	4480	2304
256	6912	9984	5120
512	15360	22016	11264
1024	33792	48128	24576

We also compared the latency of our design implementation for NTT Multiplier to the design of Chen et al. [16], Pöppelmann et al. [20], and Aysu et al. [21]. For polynomial length, n, ranging from 256 to 2048, Figure 4(c) shows a performance improvement of about 22% when compared to Chen et al. [16], about 72% when compared to Pöppelmann et al. [20], and about 78% when compared to Aysu et al [21].

V. CONCLUSION

An efficient FPGA implementation of post-quantum cryptographic hardware primitives is presented in this paper. Design choices in the implementation of the sub-modules within the PKC lead to optimal hardware cost and latency. We also presented the design implementation of an efficient polynomial multiplier. This implementation played a significant role in the efficient implementation of the entire PKC. Parameterization of the modules helped in fast prototyping and testing of the design. It also provided a means to create variably sized primitives targeting a wide range of applications.

We investigated the performance of our implementation on the Zynq-7000 FPGA. We found that our multiplier implementation has 22% $\sim$ 78% lower latency on average when compared to existing implementations [16] [20] [21]. Also, the overall hardware cost and latency is found to be optimal for a wide range of parameter sets. As for future work, we will extend our design implementation to a Somewhat Homomorphic Encryption scheme.

REFERENCES

[1] W. Knight, "Ibm raises the bar with a 50-qubit quantum computer," *Sighted at MIT Review Technology: technologyreview.com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer*, 2017.

[2] J. Hsu, "Ces 2018: Intel's 49-qubit chip shoots for quantum supremacy," *IEEE Spectrum Tech Talk*, 2018.

[3] R. Courtland, "Google aims for quantum computing supremacy [news]," *IEEE Spectrum*, vol. 54, no. 6, pp. 9–10, 2017.

[4] S. Daily. (2018) World-first quantum computer simulation of chemical bonds using trapped ions. [Online]. Available: www.sciencedaily.com/releases/2018/07/180724110028.htm

[5] R. F. Mandelbaum. (2018) This could be the best quantum computer yet. [Online]. Available: gizmodo.com/this-could-be-the-best-quantum-computer-yet-1831085617

[6] NIST. (2018) Post-quantum cryptography. [Online]. Available: csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions

[7] 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*. Springer, 2010, pp. 1–23.

[8] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-lwe cryptoprocessor," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2014, pp. 371–391.

[9] T. Poppelmann and T. Guneysu, "Area optimization of lightweight lattice-based encryption on reconfigurable hardware," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 2796–2799.

[10] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange-a new hope." in *USENIX Security Symposium*, vol. 2016, 2016.

[11] T. Oder and T. Güneysu, "Implementing the newhope-simple key exchange on low-cost fpgas," *Progress in Cryptology–LATINCRYPT*, vol. 2017, 2017.

[12] P.-C. Kuo, W.-D. Li, Y.-W. Chen, Y.-C. Hsu, B.-Y. Peng, C.-M. Cheng, and B.-Y. Yang, "High performance post-quantum key exchange on fpgas," Cryptology ePrint Archive, Report 2017/690.(2017). https://eprint. iacr. org …, Tech. Rep., 2017.

[13] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 619–631.

[14] Coindesk. (2018) Ing bank launches zero-knowledge tech for blockchain privacy. [Online]. Available: coindesk.com/ing-bank-launches-simplified-zero-knowledge-proofs-for-blockchain-privacy

[15] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila, "Post-quantum key exchange for the tls protocol from the ring learning with errors problem," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 553–570.

[16] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. Cheung, D. Pao, and I. Verbauwhede, "High-speed polynomial multiplication architecture for ring-lwe and she cryptosystems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 157–166, 2015.

[17] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, p. 34, 2009.

[18] A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede, "Constant-time discrete gaussian sampling," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1561–1571, 2018.

[19] B. Sunar, W. J. Martin, and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on computers*, vol. 56, no. 1, pp. 109–119, 2007.

[20] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2012, pp. 139–158.

[21] A. Aysu, C. Patterson, and P. Schaumont, "Low-cost and area-efficient fpga implementations of lattice-based cryptography," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2013, pp. 81–86.