IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 19 May 2023, accepted 8 July 2023, date of publication 11 July 2023, date of current version 14 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3294446

TOPICAL REVIEW

Conceptual Review on Number Theoretic Transform and Comprehensive Review on Its Implementations

ARDIANTO SATRIAWAN[1], INFALL SYAFALNI[1], RELLA MARETA[2], (Graduate Student Member, IEEE), ISA ANSHORI[1], (Member, IEEE), WERVYAN SHALANNANDA[1], AND ALEAMS BARRA[3]

[1]School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Bandung 40132, Indonesia
[2]Department of Information and Communication Engineering, Inha University, Incheon 22212, South Korea
[3]Faculty of Mathematics and Natural Sciences, Institut Teknologi Bandung, Bandung 40132, Indonesia

Corresponding author: Ardianto Satriawan (asatriawan@staff.stei.itb.ac.id)

This work was supported by the School of Electrical Engineering and Informatics, Institut Teknologi Bandung. The work of Infall Syafalni was supported by the 2023 Bank Central Asia (BCA) Innovation Awards arranged by the Lembaga Pengembangan Inovasi dan Kewirausahaan (LPIK)-Institut Teknologi Bandung (ITB).

ABSTRACT The Number Theoretic Transform (NTT) is a powerful mathematical tool that has become increasingly important in developing Post Quantum Cryptography (PQC) and Homomorphic Encryption (HE). Its ability to efficiently calculate polynomial multiplication using the convolution theorem with a quasi-linear complexity $O(n \log n)$ when implemented with Fast Fourier Transform-style algorithms has made it a key component in modern cryptography. FFT-style NTT algorithm or fast-NTT is particularly useful in lattice-based cryptography, which relies on the hardness of certain mathematical problems to ensure security. Its importance in these fields continues to grow as quantum computing technology advances and traditional encryption methods become vulnerable. In this report, we discuss the mathematical concepts of polynomial multiplications using NTT and provide a comprehensive review of the latest implementation and state-of-the-art of NTT in both PQC and HE schemes.

INDEX TERMS Number theoretic transform, post quantum cryptography, homomorphic encryption.

I. INTRODUCTION

Most of the classical cryptosystems are based on the assumption that the prime factorization of a large integer is a computationally complex problem to solve [1]. However, the assumption will no longer hold in the near future due to the recent development of quantum computer research and developments. Quantum computers can factorize large integers exponentially faster than classical computers [2]. This vulnerability has led to the need for quantum-resistant cryptosystem developments. There are five different post-quantum cryptosystems, which are lattice-based, hash-based, code-based, multivariable-based, and isogeny-based schemes [3]. Lattice-based cryptography is a promising and most researched scheme due to its balance of computing complexity, communication bandwidth, and security [4].

An effort has been initiated by the US National Institute of Standards and Technology (NIST) to standardize cryptographic algorithms that are resistant to attacks by quantum computers, which was called Post-Quantum Cryptography (PQC) Competition starting in 2016 and finalized in 2022 [5]. The lattice-based cryptography is the most proposed system, making 26 of 64 in the first round [6], 12 of 16 in the second round [7], 7 out of 15 in the third round [8], and 3 out of 4 final standardized schemes [9]. Those three lattice-based standardized cryptosystems are Dilithium [10], [11], Falcon [12], and Kyber [13], [14], [15], [16].

The bottleneck of the lattice-based cryptography implementation is its fundamental building block: modular polynomial multiplication, which is a very time-consuming operation [17]. Traditionally, it is computed by the

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamad Afendee Mohamed.

schoolbook algorithm with a quadratic complexity of $O(n^2)$. However, other alternatives exist, such as the Karatsuba algorithm [18], [19], the Toom-Cook algorithm [20], [21], and the Discrete Fourier Transform (DFT)-based algorithm [22], [23], [24]. The Karatsuba algorithm applies the divide and conquers principle to reduce the complexity by dividing the original polynomial into two parts, resulting in $O(n^{\log_2 3})$ or $O(n^{1.58})$ [25]. Toom-Cook algorithm generalizes Karatsuba algorithm by dividing into k parts, giving $O(n^{\log_k (2k-1)})$ complexity [26].

Discrete Fourier Transform (DFT) and its variant in the polynomial ring, Number Theoretic Transform (NTT) can be utilized to multiply two polynomials via convolution theorem [27], [28]. However, the classical algorithm to compute DFT or NTT is also $O(n^2)$. The fundamental difference between DFT and NTT is the ring they use to transform the polynomial. DFT uses a complex ring with a twiddle factor of $e^{-2\pi j/n}$ while NTT uses an integer polynomial ring with a twiddle factor of its n-th root of unity. The only use of integers makes NTT popular among researchers because there is no need to implement complicated schemes such as fixed-point or floating-point arithmetic architecture. This advantage also eliminates the precision problem that may arise from implementing such architectures [29].

Many optimized versions of DFT have been proposed in the past few decades due to their prominent use in signal and image processing. The most widely used fast algorithm is Fast Fourier Transform (FFT) which Gauss first proposed in 1805 [30]. It gained widespread attention in the 1960s when Cooley-Tukey [23] and Gentleman-Sande [24] published their works, giving their infamous name for the CT and GS butterflies architecture for FFT. The FFT has a quasilinear complexity of $O(n \log n)$, which gives a massive advantage over other methods, especially when calculating higher-degree polynomial multiplications. NTT is also a DFT version, so one can apply FFT algorithms to calculate NTT [31].

However, using NTT also has limitations: it requires very specific parameters. Implementing FFT algorithms requires the array lengths n to be a power of two – in other words, the polynomials need to have a $2^k - 1$ degree [32]. It also only works on a specific prime modulus. Positive-wrapped convolution (PWC)-based NTT requires the prime modulus q to have a primitive n-th root of unity in the $\mathbb{Z}_q$ ring. Moreover, negative-wrapped convolution (NWC)-based NTT needs an additional $2n$-th root of unity [33].

The parameter requirements of NTT make it not always available to use in lattice-based cryptosystems. Out of three standardized PQC schemes, while Dilithium and Falcon can apply PWC-based and NWC-based NTT, Kyber can only use PWC-based NTT due to its chosen parameters. In the other finalists' schemes: NTRU and Saber, NTT can not be used due to the power-of-two modulus and the chosen ring, respectively [33]. However, many researchers are working on making workarounds to implement NTT on such systems using various mathematical techniques, such as the Residue Number Systems (RNS) and the Chinese Remainder Theorem (CRT) [34], [35].

NTT is also important in Homomorphic Encryption (HE) schemes such as Brakerski-Fan-Vercauteren (BFV) [36], BGV (Brakerski-Gentry-Vaikuntanathan) [37], [38], and CKKS (Cheon-Kim-Kim-Song) [39] based on the Ring Learning With Errors (RLWE) problem. In the BFV and BGV schemes, NTT performs the modulus-switching operation to reduce the noise in the encrypted data. In the CKKS homomorphic encryption scheme, NTT performs the ''relinearization'' operation, which reduces the size of the ciphertexts after multiplication operations [36], [37], [38], [39]. Microsoft SEAL is one of the most prominent libraries implementing the aforementioned schemes [40], [41]. The noticeable difference between the PQC and HE schemes is the modulus size. While PQC schemes usually use a small number as their modulus, HE schemes use a large number, which makes the implementation techniques vastly different between the two schemes.

Most of the NTT implementation reports briefly introduce NTT and recent literature reviews. However, those reports focus on their implementation techniques of NTT in the various platforms and do not provide a comprehensive understanding of the NTT concepts. This motivates us to briefly introduce NTT concepts and summarize the state of the arts of NTT implementations in the PQC and HE schemes. We summarize the contribution of our works as follows:

1) We briefly introduce the basic concepts of linear, cyclic, and negacyclic convolutions via traditional schoolbook algorithms, traditional NTT, and FFT-like versions of NTT. While other literature briefly introduces the concepts, they are scattered everywhere. They require significant effort to learn, especially for those who begin researching the area and come from the implementation side.
2) We provide consistent toy examples through different concepts and algorithms to further enhance the conceptual understanding of the NTT. However, the focus of our report is the implementation of NTT. For the mathematical understanding of NTT, [33] provides a comprehensive conceptual explanation of the topic.
3) We summarize and provide a comprehensive review of the recent research on the NTT implementations for PQC schemes in various platforms such as FPGA, ASIC, CPU, and GPU.
4) Similarly, we also summarize and provide a comprehensive review of NTT implementations for HE schemes, which are usually a combination of RNS and CRT.

We hope that our report provides researchers in relevant fields with a general understanding of NTT from the implementation side of view and also shows the state-of-the-art of NTT implementations in various architectures.

The rest of the paper is organized as follows. Section II discusses the fundamental mathematical definitions and basic concepts of convolutions in polynomial rings. Section III explains convolutions based on the Number Theoretic Transform. Section IV explains FFT-like algorithms in calculating NTT. Section V reviews the current research works of NTT implementations in Post Quantum Cryptography (PQC) scheme. Section VI reviews the current research works of NTT implementations in the Homomorphic Encryption (HE) scheme. Finally, Section VII concludes the paper and discusses possible future works.

II. PRELIMINARIES: SCHOOLBOOK CONVOLUTIONS

This section briefly explains the definition of linear, cyclic, and negacyclic convolutions between polynomials with integer coefficients to show their basic concepts and differences. We also provide simple and consistent toy examples throughout the section to clarify how different concepts work. In this section, we assume the modulus, q, is large enough so that the arithmetic calculations do not cause integer overflows.

A. POLYNOMIAL MULTIPLICATION AND LINEAR CONVOLUTION

Definition 2.1: Suppose that $G(x)$ and $H(x)$ are polynomials of degree $n-1$ in the ring $\mathbb{Z}_q[x]$ where $q \in \mathbb{Z}$ and x is the polynomial variable, a ***polynomial multiplication*** *of $G(x)$ and $H(x)$ is defined as*:

$$\begin{aligned} Y(x) &= G(x) \cdot H(x) \\ &= \sum_{k=0}^{2(n-1)} y_k x^k \end{aligned} \tag{1}$$

where $y_k = \sum_{i=0}^{k} g_i h_{k-i} \mod q$, $\boldsymbol{g}$ and $\boldsymbol{h}$ are the polynomial coefficients of $G(x)$ and $H(x)$ respectively.

Polynomial multiplication is equivalent to a discrete **linear convolution** between the coefficients' vectors $\boldsymbol{g}$ and $\boldsymbol{h}$ [42].

$$\boldsymbol{y}[k] = (\boldsymbol{g} * \boldsymbol{h})[k] = \sum_{i=0}^{k} \boldsymbol{g}[i]\boldsymbol{h}[k-i] \tag{2}$$

Example 2.1: Let $G(x) = 1+2x+3x^2+4x^3$ and $H(x) = 5+6x+7x^2+8x^3$ or in vector notation $\boldsymbol{g} = [1,2,3,4]$ and $\boldsymbol{h} = [5,6,7,8]$. The result of the linear convolution is $Y(x) = 5+16x+34x^2+60x^3+61x^4+52x^5+32x^6$ or $\boldsymbol{y} = [5,16,34,60,61,52,32]$.

Figure 1 shows the schoolbook method of how a typical polynomial multiplication or linear convolution is done. This traditional multiplication algorithm has a $O(n^2)$ complexity.

The algorithm can be implemented in many mathematical programming libraries, such as MATLAB's `conv` [43] and Numpy's `convolve` [44] with integer array inputs combined with modular arithmetic operations.

B. CYCLIC CONVOLUTION

Definition 2.2: Suppose that $G(x)$ and $H(x)$ are polynomials of degree $n-1$ in the quotient ring $\mathbb{Z}_q[x]/(x^n-1)$ where $q \in \mathbb{Z}$. A ***cyclic convolution*** *or* ***positive wrapped convolution****, $PWC(x)$ is defined as*:

$$PWC(x) = \sum_{k=0}^{n-1} c_k x^k \tag{3}$$

where $c_k = \sum_{i=0}^{k} g_i h_{k-i} + \sum_{i=k+1}^{n-1} g_i h_{k+n-i} \mod q$. If $Y(x)$ is the result of their linear convolution in the ring $\mathbb{Z}_q[x]$, it also can be defined as

$$PWC(x) = Y(x) \bmod (x^n - 1) \tag{4}$$

Traditional and schoolbooks method to calculate a cyclic convolution is through a polynomial multiplication, as shown in Example 2.1, followed by a long division. The method has $O(n^2)$ complexity.

Example 2.2: Let $G(x) = 1+2x+3x^2+4x^3$ and $H(x) = 5+6x+7x^2+8x^3$ or in vector notation $\boldsymbol{g} = [1,2,3,4]$ and $\boldsymbol{h} = [5,6,7,8]$. The result of the cyclic convolution is $PWC(x) = 66+68x+66x^2+60x^3$ or $[66,68,66,60]$.

Figure 2 shows how schoolbook long division is used to calculate a cyclic convolution with the dividend as the linear convolution result of $G(x)$ and $H(x)$. The remainder of the long division algorithm is the cyclic convolution result. Notice that we present the result sorted in increasing power in Example 2.2.

$$\begin{array}{r} 1 + 2x + 3x^2 + 4x^3 \\ 5 + 6x + 7x^2 + 8x^3 \\ \hline 8x^3 + 16x^4 + 24x^5 + 32x^6 \quad \times \\ 7x^2 + 14x^3 + 21x^4 + 28x^5 \\ 6x + 12x^2 + 18x^3 + 24x^4 \\ 5 + 10x + 15x^2 + 20x^3 \\ \hline 5 + 16x + 34x^2 + 60x^3 + 61x^4 + 52x^5 + 32x^6 \quad + \end{array}$$

FIGURE 1. Schoolbook method for polynomial multiplication or linear convolution.

$$\begin{array}{r|l} & 32x^2 + 52x + 61 \\ \hline x^4 - 1 & 32x^6 + 52x^5 + 61x^4 + 60x^3 + 34x^2 + 16x + 5 \\ & 32x^6 + 0x^5 + 0x^4 + 0x^3 - 32x^2 \quad - \\ \hline & 52x^5 + 61x^4 + 60x^3 + 66x^2 + 16x + 5 \\ & 52x^5 + 0x^4 + 0x^3 + 0x^2 - 52x \quad - \\ \hline & 61x^4 + 60x^3 + 66x^2 + 68x + 5 \\ & 61x^4 + 0x^3 + 0x^2 + 0x - 61 \quad - \\ \hline & 60x^3 + 66x^2 + 68x + 66 \end{array}$$

FIGURE 2. Schoolbook method for positively wrapped modular polynomial multiplication or cyclic convolution.

The MATLAB function `cconv` [45] can calculate a cyclic convolution using integer array inputs and modular arithmetic operations. Notice that the result of cyclic convolution, unlike linear convolution, has a length of n instead of $2n-1$.

C. NEGACYCLIC CONVOLUTION

Definition 2.3: Suppose that $G(x)$ and $H(x)$ are polynomials of degree $n-1$ in the quotient ring $\mathbb{Z}[x]/(x^n+1)$ where $q \in \mathbb{Z}$. A ***negacyclic convolution*** *or* ***negative wrapped convolution****, $NWC(x)$ is defined as*:

$$NWC(x) = \sum_{k=0}^{n-1} c_k x^k \tag{5}$$

where $c_k = \sum_{i=0}^{k} g_i h_{k-i} - \sum_{i=k+1}^{n-1} g_i h_{k+n-i} \mod q$. If $Y(x)$ is the result of their linear convolution in the ring $\mathbb{Z}[x]$, it also can be defined as

$$NWC(x) = Y(x) \bmod (x^n+1) \tag{6}$$

Example 2.3: Let $G(x) = 1+2x+3x^2+4x^3$ and $H(x) = 5+6x+7x^2+8x^3$ or in vector notation $\boldsymbol{g} = [1,2,3,4]$ and $\boldsymbol{h} = [5,6,7,8]$. The result of the negacyclic convolution is $NWC(x) = -56-36x+2x^2+60x^3$ or $[-56,-36,2,60]$.

Figure 3 shows how schoolbook long division calculates a negacyclic convolution, the remainder of the division.

$$
\begin{array}{r|l}
 & 32x^2+52x+61 \\
\hline
x^4+1 & 32x^6+52x^5+61x^4+60x^3+34x^2+16x+5 \\
 & \underline{32x^6+\ 0x^5+\ 0x^4+\ 0x^3+32x^2} \quad - \\
 & \quad 52x^5+61x^4+60x^3+66x^2+16x+5 \\
 & \quad \underline{52x^5+\ 0x^4+\ 0x^3+\ 0x^2+52x} \quad - \\
 & \quad\quad 61x^4+60x^3+66x^2+68x+5 \\
 & \quad\quad \underline{61x^4+\ 0x^3+\ 0x^2+\ 0x+61} \quad - \\
 & \quad\quad\quad 60x^3+2x^2-36x-56
\end{array}
$$

FIGURE 3. Schoolbook method for negatively wrapped modular polynomial multiplication or negacyclic convolution.

Note that the only difference between cyclic and negacyclic convolution is the divisor. The cyclic convolution uses x^n-1 while the negacyclic convolution uses x^n+1.

Those schoolbook algorithms have $O(n^2)$ complexity. Many efforts have been tried to reduce their complexities by dividing the multiplier and multiplicand into several parts [18], [19], [20], [21] or by parallelizing the algorithm on the implementation side [46]. However, those efforts are not scalable as the polynomial degree grows higher.

III. NTT-BASED CONVOLUTIONS

In this section, we present the basic of NTT-based convolutions. Many researchers do not differentiate the term NTT and FFT-based algorithms to calculate NTT, which creates confusion when understanding the topic. This report refers to the transformation itself as **NTT** and the FFT-like algorithms as **fast-NTT**, which are explained in Section IV. The classical NTT has quadratic complexity of $O(n^2)$ when computed directly, while fast-NTT algorithms have a more efficient quasi-linear complexity $O(n \log n)$.

A. PRIMITIVE n-TH ROOT OF UNITY

Definition 3.1: Let $\mathbb{Z}_q$ be an integer ring modulo q, and $n-1$ is the polynomial degree of $G(x)$ and $H(x)$. Such rings have a multiplicative identity (unity) of 1. Define ω as ***primitive n-th root of unity*** *in $\mathbb{Z}_q$ if and only if*:

$$\omega^n \equiv 1 \mod q \tag{7}$$

and

$$\omega^k \not\equiv 1 \mod q \tag{8}$$

for $k < n$.

One thing to note is that the primitive $n-th$ root of unity in a ring $\mathbb{Z}_q$ might not be unique. We show the following example for $q = 7681$, used in Kyber in Rounds 1 and 2 of the NIST-PQC Competition [13], [15], however, in our toy example we show for $n = 4$ instead of $n = 256$.

Example 3.1: In a ring $\mathbb{Z}_{7681}$ and $n = 4$, the 4-th root of unity which satisfy the condition $\omega^4 \equiv 1 \mod 7681$ are $\{3383, 4298, 7680\}$. Out of three roots, 7680 is ***not a primitive*** *4-th root of unity, as there exist $k = 2 < n$ that satisfy $\omega^2 \equiv 1 \mod 7681$. Therefore $\omega = 3383$ or $\omega = 4298$ are the primitive 4-th root of unity in $\mathbb{Z}_{7681}$.*

The value of ω will be important in calculating NTT and positive-wrapped convolution. Calculating the ω of a ring with a large number modulus q is tricky and tedious. One alternative library that provides a function to calculate ω is Sympy via the function `nthroot_mod` [47].

B. NTT-BASED POSITIVE-WRAPPED CONVOLUTION

This section explains the definition of Number Theoretic Transform (NTT) and its inverse (INTT) based on n-th root of unity, ω. The NTT of a polynomial does not have any physical meaning, unlike Discrete Fourier Transform (DFT) which represents a signal in the frequency domain. However, NTT preserves one of the important properties of DFT: the convolution theorem, which is valuable in calculating polynomial multiplication.

1) NUMBER THEORETIC TRANSFORM BASED ON ω

Definition 3.2: The Number Theoretic Transform (NTT) of a vector of polynomial coefficients $\boldsymbol{a}$ is defined as $\hat{\boldsymbol{a}} = NTT(\boldsymbol{a})$, where:

$$\hat{\boldsymbol{a}}_j = \sum_{i=0}^{n-1} \omega^{ij} a_i \mod q \tag{9}$$

and $j = 0, 1, 2, \ldots, n-1$

Example 3.2: Let $G(x) = 1+2x+3x^2+4x^3$ or in vector notation $\boldsymbol{g} = [1,2,3,4]$. We can infer that $n = 4$. Suppose we work in the ring $\mathbb{Z}_{7681}$ and ω is its primitive n-th root of unity. The NTT of $\boldsymbol{g}$, $\hat{\boldsymbol{g}}$, can be calculated by the following matrix multiplication:

$$\hat{\boldsymbol{g}} = \begin{bmatrix} \omega^{0\times0} & \omega^{0\times1} & \omega^{0\times2} & \omega^{0\times3} \\ \omega^{1\times0} & \omega^{1\times1} & \omega^{1\times2} & \omega^{1\times3} \\ \omega^{2\times0} & \omega^{2\times1} & \omega^{2\times2} & \omega^{2\times3} \\ \omega^{3\times0} & \omega^{3\times1} & \omega^{3\times2} & \omega^{3\times3} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Notice that the power of ω is the multiplication between the row and column numbers. As ω is the n-root of unity, $\omega^k = \omega^{(k \bmod n)}$ for $k > n$. Thus:

$$\hat{\boldsymbol{g}} = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\hat{\boldsymbol{g}} = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^0 & \omega^2 \\ \omega^0 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

From Example 3.1 we obtained one of the n-th roots of unity in $\mathbb{Z}_{7681}$ is $\omega = 3383$. Substituting into the equation:

$$\hat{\boldsymbol{g}} = \begin{bmatrix} 3383^0 & 3383^0 & 3383^0 & 3383^0 \\ 3383^0 & 3383^1 & 3383^2 & 3383^3 \\ 3383^0 & 3383^2 & 3383^0 & 3383^2 \\ 3383^0 & 3383^3 & 3383^2 & 3383^1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\hat{\boldsymbol{g}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 3383 & 7680 & 4298 \\ 1 & 7680 & 1 & 7680 \\ 1 & 4298 & 7680 & 3383 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\hat{\boldsymbol{g}} = \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

Therefore, the NTT($\boldsymbol{g}$) = [10, 913, 7679, 6764] in $\mathbb{Z}_{7681}$.

Example 3.3: Let $H(x) = 5 + 6x + 7x^2 + 8x^3$ or in vector notation $\boldsymbol{g} = [5, 6, 7, 8]$ in the ring $\mathbb{Z}_{7681}$ and $\omega = 3383$. The NTT of $\boldsymbol{h}$ is:

$$\hat{\boldsymbol{h}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 3383 & 7680 & 4298 \\ 1 & 7680 & 1 & 7680 \\ 1 & 4298 & 7680 & 3383 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 26 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

Therefore, the NTT($\boldsymbol{h}$) = [26, 913, 7679, 6764] in $\mathbb{Z}_{7681}$.

Note that the NTT of a particular polynomial is not always unique. It depends on the choice of ω. The NTT result of Example 3.2 and 3.3 will differ if one uses $\omega = 4298$ instead of $\omega = 3383$.

2) INVERSE NUMBER THEORETIC TRANSFORM BASED ON ω

Definition 3.3: The Inverse of Number Theoretic Transform (INTT) of an NTT vector $\hat{\boldsymbol{a}}$ is defined as $\boldsymbol{a} = INTT(\hat{\boldsymbol{a}})$, where:

$$\boldsymbol{a}_i = n^{-1} \sum_{j=0}^{n-1} \omega^{-ij} \hat{a}_j \mod q \tag{10}$$

and $j = 0, 1, 2, \ldots, n-1$

Note that the INTT has a very similar formula to NTT. The only differences are ω replaced by its inverse in $\mathbb{Z}_q$ and a n^{-1} scaling factor. It always holds that $\boldsymbol{a} = \text{INTT}(\text{NTT}(\boldsymbol{a}))$.

Example 3.4: Given NTT($\boldsymbol{g}$) = $\hat{\boldsymbol{g}}$ = [10, 913, 7679, 6764] in $\mathbb{Z}_{7681}$ and $\omega = 3383$. We can calculate the inverse of ω, $\omega^{-1} = 4298$ and the scaling factor $n^{-1} = 5761$. One can calculate the INTT(NTT($\hat{\boldsymbol{g}}$)) by the following matrix multiplication:

$$g = n^{-1} \begin{bmatrix} \omega^{-0\times0} & \omega^{-0\times1} & \omega^{-0\times2} & \omega^{-0\times3} \\ \omega^{-1\times0} & \omega^{-1\times1} & \omega^{-1\times2} & \omega^{-1\times3} \\ \omega^{-2\times0} & \omega^{-2\times1} & \omega^{-2\times2} & \omega^{-2\times3} \\ \omega^{-3\times0} & \omega^{-3\times1} & \omega^{-3\times2} & \omega^{-3\times3} \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

$$g = n^{-1} \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ \omega^0 & \omega^{-2} & \omega^{-4} & \omega^{-6} \\ \omega^0 & \omega^{-3} & \omega^{-6} & \omega^{-9} \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

$$g = n^{-1} \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ \omega^0 & \omega^{-2} & \omega^{-0} & \omega^{-2} \\ \omega^0 & \omega^{-3} & \omega^{-2} & \omega^{-1} \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

$$g = 5761 \begin{bmatrix} 4298^0 & 4298^0 & 4298^0 & 4298^0 \\ 4298^0 & 4298^1 & 4298^2 & 4298^3 \\ 4298^0 & 4298^2 & 4298^0 & 4298^2 \\ 4298^0 & 4298^3 & 4298^2 & 4298^1 \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix}$$

$$g = 5761 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4298 & 7680 & 3383 \\ 1 & 7680 & 1 & 7680 \\ 1 & 3383 & 7680 & 4298 \end{bmatrix} \begin{bmatrix} 10 \\ 913 \\ 7679 \\ 6764 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Therefore, the $\boldsymbol{g} = [1, 2, 3, 4]$, which is the initial polynomial coefficients given in Example 3.2

Example 3.5: Given NTT($\boldsymbol{g}$) = $\hat{\boldsymbol{h}}$ = [26, 913, 7679, 6764] in $\mathbb{Z}_{7681}$ and $\omega = 3383$. We can similarly calculate the INTT to the previous example:

$$h = 5761 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4298 & 7680 & 3383 \\ 1 & 7680 & 1 & 7680 \\ 1 & 3383 & 7680 & 4298 \end{bmatrix} \begin{bmatrix} 26 \\ 913 \\ 7679 \\ 6764 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

Therefore, the $\boldsymbol{h} = [5, 6, 7, 8]$, which is the initial polynomial coefficients given in Example 3.3

3) USING NTT TO CALCULATE POSITIVE-WRAPPED CONVOLUTIONS

Because NTT is a variant of DFT in the polynomial ring. One can apply DFT's convolution theorem to calculate positive-wrapped convolution [27], [28]:

Proposition 3.1: Let $\boldsymbol{a}$ and $\boldsymbol{b}$ are the multiplicands' vectors of polynomial coefficients. The positive-wrapped convolution of $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{c}$ can be calculated by:

$$\boldsymbol{c} = INTT(NTT(\boldsymbol{a}) \circ NTT(\boldsymbol{b})) \tag{11}$$

where $\circ$ is an element-wise vector multiplication in $\mathbb{Z}_q$.

Example 3.6: Let $\boldsymbol{g} = [1, 2, 3, 4]$ and $\boldsymbol{h} = [5, 6, 7, 8]$. From Example 3.2 and 3.3, we know that the NTT of them in in $\mathbb{Z}_{7681}$ are $\hat{\boldsymbol{g}} = [10, 913, 7679, 6764]$ and $\hat{\boldsymbol{h}} = [10, 913, 7679, 6764]$ when $\omega = 3383$. We can calculate

their positive-wrapped convolution by:

$$INTT(\begin{bmatrix}10\\913\\7679\\6764\end{bmatrix} \circ \begin{bmatrix}26\\913\\7679\\6764\end{bmatrix}) = INTT(\begin{bmatrix}260\\4021\\4\\3660\end{bmatrix}) = 5761 \begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 4298 & 7680 & 3383\\ 1 & 7680 & 1 & 7680\\ 1 & 3383 & 7680 & 4298\end{bmatrix} \begin{bmatrix}260\\4021\\4\\3660\end{bmatrix} = \begin{bmatrix}66\\68\\66\\60\end{bmatrix}$$

Therefore, their positive-wrapped convolution is $[66, 68, 66, 60]$*, the same result as calculated by schoolbook multiplication and long division in Example 2.2.*

While positive-wrapped convolution, commonly known as cyclic convolution, is useful, its implementation is primarily outside the cryptography domain. One such example is the implementation of Schönhage-Strassen algorithm [48] for large integer multiplication. However, in the context of PQC and HE, the chosen ring is mostly $\mathbb{Z}_q[n]/(x^n + 1)$ instead of $\mathbb{Z}_q[n]/(x^n - 1)$. One must calculate the polynomial multiplications via the negative-wrapped convolution in such rings.

C. PRIMITIVE 2n-TH ROOT OF UNITY

To calculate negative-wrapped convolution, one needs the primitive $2n$-th root of unity, ψ.

Definition 3.4: Let $\mathbb{Z}_q$ *be an integer ring modulo* q*, and* $n-1$ *is the polynomial degree of* $G(x)$ *and* $H(x)$ *and* ω *is its primitive* n*-th root of unity. Define* ψ *as the primitive* $2n$*-th root of unity if and only if*:

$$\psi^2 \equiv \omega \mod q \tag{12}$$

and

$$\psi^n \equiv -1 \mod q \tag{13}$$

Example 3.7: In a ring $\mathbb{Z}_{7681}$ *and* $n = 4$*, when* $\omega = 3383$*, the value of* ψ *can be 1925 or 5756 as* $1925^2 = 5756^2 \equiv 3383 \bmod 7681$ *and* $1925^4 = 5756^4 = 7680 \equiv -1 \bmod 7681$*. Therefore, one can choose the value of* $\psi = 1925$ *or* $\psi = 5756$*.*

D. NTT-BASED NEGATIVE-WRAPPED CONVOLUTION

This section explains the definition of Number Theoretic Transform (NTT) and its inverse (INTT) based on $2n$-th root of unity, ψ, and how to utilize them to calculate negative-wrapped or negacyclic convolution.

1) NUMBER THEORETIC TRANSFORM BASED ON ψ

Definition 3.5: The Negative-Wrapped Number Theoretic Transform (NTT^{ψ}*) of a vector of polynomial coefficients* $\boldsymbol{a}$ *is defined as* $\hat{\boldsymbol{a}} = NTT^{\psi}(\boldsymbol{a})$*, where*:

$$\hat{\boldsymbol{a}}_j = \sum_{i=0}^{n-1} \psi^i \omega^{ij} a_i \mod q \tag{14}$$

and $j = 0, 1, 2, \ldots, n-1$*. As* $\psi^2 \equiv \omega \bmod q$*, we can substitute* $\omega = \psi^2$ *to equation (14):*

$$\hat{\boldsymbol{a}}_j = \sum_{i=0}^{n-1} \psi^{2ij+i} a_i \mod q \tag{15}$$

Example 3.8: Let $\boldsymbol{g} = [1, 2, 3, 4]$*,* $n = 4$ *and* $\psi = 1925$ *in the ring* $\mathbb{Z}_{7681}$*. The* $NTT^{\psi}(\boldsymbol{g}) = \hat{\boldsymbol{g}}$*, can be calculated by the following matrix multiplication*:

$$\hat{\boldsymbol{g}} = \begin{bmatrix}\psi^{2(0\times0)+0} & \psi^{2(0\times1)+1} & \psi^{2(0\times2)+2} & \psi^{2(0\times3)+3}\\ \psi^{2(1\times0)+0} & \psi^{2(1\times1)+1} & \psi^{2(1\times2)+2} & \psi^{2(1\times3)+3}\\ \psi^{2(2\times0)+0} & \psi^{2(2\times1)+1} & \psi^{2(2\times2)+2} & \psi^{2(2\times3)+3}\\ \psi^{2(3\times0)+0} & \psi^{2(3\times1)+1} & \psi^{2(3\times2)+2} & \psi^{2(3\times3)+3}\end{bmatrix} \begin{bmatrix}1\\2\\3\\4\end{bmatrix}$$

$$\hat{\boldsymbol{g}} = \begin{bmatrix}\psi^0 & \psi^1 & \psi^2 & \psi^3\\ \psi^0 & \psi^3 & \psi^6 & \psi^9\\ \psi^0 & \psi^5 & \psi^{10} & \psi^{15}\\ \psi^0 & \psi^7 & \psi^{14} & \psi^{21}\end{bmatrix} \begin{bmatrix}1\\2\\3\\4\end{bmatrix} = \begin{bmatrix}\psi^0 & \psi^1 & \psi^2 & \psi^3\\ \psi^0 & \psi^3 & \psi^6 & \psi^1\\ \psi^0 & \psi^5 & \psi^2 & \psi^7\\ \psi^0 & \psi^7 & \psi^6 & \psi^5\end{bmatrix} \begin{bmatrix}1\\2\\3\\4\end{bmatrix}$$

$$\hat{\boldsymbol{g}} = \begin{bmatrix}1925^0 & 1925^1 & 1925^2 & 1925^3\\ 1925^0 & 1925^3 & 1925^6 & 1925^1\\ 1925^0 & 1925^5 & 1925^2 & 1925^7\\ 1925^0 & 1925^7 & 1925^6 & 1925^5\end{bmatrix} \begin{bmatrix}1\\2\\3\\4\end{bmatrix}$$

$$\hat{\boldsymbol{g}} = \begin{bmatrix}1 & 1925 & 3383 & 6468\\ 1 & 6468 & 4298 & 1925\\ 1 & 5756 & 3383 & 1213\\ 1 & 1213 & 4298 & 5756\end{bmatrix} \begin{bmatrix}1\\2\\3\\4\end{bmatrix} = \begin{bmatrix}1467\\2807\\3471\\7621\end{bmatrix}$$

Therefore, the $NTT^{\psi}(\boldsymbol{g}) = [1467, 2807, 3471, 7621]$ *when* $\psi = 1925$ *in* $\mathbb{Z}_{7681}$*.*

Example 3.9: Let $\boldsymbol{h} = [5, 6, 7, 8]$*,* $n = 4$ *and* $\psi = 1925$ *in the ring* $\mathbb{Z}_{7681}$*. The* $NTT^{\psi}(\boldsymbol{h}) = \hat{\boldsymbol{h}}$*, can be calculated similarly by the following matrix multiplication*:

$$\hat{\boldsymbol{h}} = \begin{bmatrix}1 & 1925 & 3383 & 6468\\ 1 & 6468 & 4298 & 1925\\ 1 & 5756 & 3383 & 1213\\ 1 & 1213 & 4298 & 5756\end{bmatrix} \begin{bmatrix}5\\6\\7\\8\end{bmatrix} = \begin{bmatrix}2489\\7489\\6478\\6607\end{bmatrix}$$

Therefore, the $NTT^{\psi}(\boldsymbol{h}) = [2489, 7489, 6478, 6607]$*.*

2) INVERSE NUMBER THEORETIC TRANSFORM BASED ON ψ

Definition 3.6: The Negative-Wrapped Inverse of Number Theoretic Transform (INTT) of an NTT vector $\hat{\boldsymbol{a}}$ *is defined as* $\boldsymbol{a} = INTT^{\psi^{-1}}(\hat{\boldsymbol{a}})$*, where*:

$$\boldsymbol{a}_i = n^{-1} \sum_{j=0}^{n-1} \psi^{-j} \omega^{-ij} \hat{a}_j \mod q \tag{16}$$

and $i = 0, 1, 2, \ldots, n-1$*. Substituting* $\omega = \psi^2$ *yields*:

$$\boldsymbol{a}_i = n^{-1} \sum_{j=0}^{n-1} \psi^{-(2ij+j)} \hat{a}_j \mod q \tag{17}$$

Note that the differences between NTT^{ψ} and INTT^{ψ} are the scaling factor n^{-1}, the replacement of ψ by ψ^{-1}, and the transpose of the exponents of ψ matrix.

Example 3.10: Let $NTT^{\psi}(\boldsymbol{g}) = \hat{\boldsymbol{g}} = [1467, 2807, 3471, 7621]$ *and* $\psi = 1925$ *in the ring* $\mathbb{Z}_{7681}$. *Note that* $\psi^{-1} = 1213$ *and* $n^{-1} = 5761$. *The vector* $\boldsymbol{g}$ *can be calculated by the following matrix multiplication*:

$$\boldsymbol{g} = n^{-1}\begin{bmatrix} \psi^{-0} & \psi^{-0} & \psi^{-0} & \psi^{-0} \\ \psi^{-1} & \psi^{-3} & \psi^{-5} & \psi^{-7} \\ \psi^{-2} & \psi^{-6} & \psi^{-10} & \psi^{-14} \\ \psi^{-3} & \psi^{-9} & \psi^{-15} & \psi^{-21} \end{bmatrix}\begin{bmatrix} 1467 \\ 2807 \\ 3471 \\ 7621 \end{bmatrix}$$

$$\boldsymbol{g} = n^{-1}\begin{bmatrix} \psi^{-0} & \psi^{-0} & \psi^{-0} & \psi^{-0} \\ \psi^{-1} & \psi^{-3} & \psi^{-5} & \psi^{-7} \\ \psi^{-2} & \psi^{-6} & \psi^{-2} & \psi^{-6} \\ \psi^{-3} & \psi^{-1} & \psi^{-7} & \psi^{-5} \end{bmatrix}\begin{bmatrix} 1467 \\ 2807 \\ 3471 \\ 7621 \end{bmatrix}$$

$$\boldsymbol{g} = 5761\begin{bmatrix} 1213^{0} & 1213^{0} & 1213^{0} & 1213^{0} \\ 1213^{1} & 1213^{3} & 1213^{5} & 1213^{7} \\ 1213^{2} & 1213^{6} & 1213^{2} & 1213^{6} \\ 1213^{3} & 1213^{1} & 1213^{7} & 1213^{5} \end{bmatrix}\begin{bmatrix} 1467 \\ 2807 \\ 3471 \\ 7621 \end{bmatrix}$$

$$\boldsymbol{g} = 5761\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1213 & 5756 & 6468 & 1925 \\ 4298 & 3383 & 4298 & 3383 \\ 5756 & 1213 & 1925 & 6468 \end{bmatrix}\begin{bmatrix} 1467 \\ 2807 \\ 3471 \\ 7621 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Therefore $\boldsymbol{g} = [1, 2, 3, 4]$.

Example 3.11: Let $NTT^{\psi}(\boldsymbol{h}) = \hat{\boldsymbol{h}} = [2489, 7489, 6478, 6607]$ *and* $\psi = 1925$ *in the ring* $\mathbb{Z}_{7681}$. *The vector* $\boldsymbol{h}$ *can be calculated by the following matrix multiplication*:

$$\boldsymbol{h} = 5761\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1213 & 5756 & 6468 & 1925 \\ 4298 & 3383 & 4298 & 3383 \\ 5756 & 1213 & 1925 & 6468 \end{bmatrix}\begin{bmatrix} 2489 \\ 7489 \\ 6478 \\ 6607 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

Therefore, the $\boldsymbol{h} = [5, 6, 7, 8]$.

3) USING NTT^{ψ} TO CALCULATE NEGATIVE-WRAPPED CONVOLUTIONS

Like its positive-wrapped version, the negative-wrapped NTT can evaluate the negative-wrapped convolutions, commonly referred to as negacyclic convolutions.

Proposition 3.2: Let $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *are the multiplicands' vectors of polynomial coefficients. The negative-wrapped convolution of* $\boldsymbol{a}$ *and* $\boldsymbol{b}$, $\boldsymbol{c}$ *can be calculated by*:

$$\boldsymbol{c} = INTT^{\psi^{-1}}(NTT^{\psi}(\boldsymbol{a}) \circ NTT^{\psi}(\boldsymbol{b})) \tag{18}$$

where $\circ$ *is an element-wise vector multiplication in* $\mathbb{Z}_q$.

Example 3.12: Let $\boldsymbol{g} = [1, 2, 3, 4]$ *and* $\boldsymbol{h} = [5, 6, 7, 8]$. *From Example 3.8 and 3.9, we know that the* NTT^{ψ} *of them in in* $\mathbb{Z}_{7681}$ *are* $\hat{\boldsymbol{g}} = [1467, 2807, 3471, 7621]$ *and* $\hat{\boldsymbol{h}} = [2489, 7489, 6478, 6607]$ *when* $\psi = 1925$. *We can calculate their negative-wrapped convolution by*:

$$INTT(\begin{bmatrix} 1467 \\ 2807 \\ 3471 \\ 7621 \end{bmatrix} \circ \begin{bmatrix} 2489 \\ 7489 \\ 6478 \\ 6607 \end{bmatrix})$$

$$= INTT(\begin{bmatrix} 2888 \\ 6407 \\ 2851 \\ 2992 \end{bmatrix})$$

$$= 5761\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1213 & 5756 & 6468 & 1925 \\ 4298 & 3383 & 4298 & 3383 \\ 5756 & 1213 & 1925 & 6468 \end{bmatrix}\begin{bmatrix} 2888 \\ 6407 \\ 2851 \\ 2992 \end{bmatrix} = \begin{bmatrix} 7625 \\ 7645 \\ 2 \\ 60 \end{bmatrix}$$

Therefore, $[7625, 7645, 2, 60]$ – *or when written with negative numbers* $[-56, -36, 2, 60]$ *is their negacyclic convolution, the same result as calculated by schoolbook multiplication and long division in Example 2.3*

E. THE CHOICE OF MODULUS

To make NTT transformation available, the modulus q has to satisfy the following requirements:

1) The n-th root of unity ω exists in ring $\mathbb{Z}_q$. The existence of ω enables one to utilize NTT to perform positive-wrapped convolutions.
2) Furthermore, the $2n$-th root of unity ψ exists in ring $\mathbb{Z}_q$ to make negative-wrapped convolutions work.

The modulus q has to satisfy the following theorem to guarantee that ω exists [27], [29], [49]:

Theorem 3.1: If q *is prime, then* n *must divide* $q - 1$. *If* q *is composite such that*:

$$q = {q_1}^{m_1} \cdot {q_2}^{m_2} \cdot {q_3}^{m_3} \ldots {q_k}^{m_k}$$

then n *must divide the greatest common divisor (GCD) of* $(q_1 - 1, q_2 - 1, q_3 - 1, \ldots, q_k - 1)$.

However, while Theorem 3.1 guarantees the existence of ω does not guarantee the existence of ψ. To guarantee the existence of ψ in $\mathbb{Z}_q$:

Theorem 3.2: If q *is prime, then* $2n$ *must divide* $q - 1$. *If* q *is composite such that*:

$$q = {q_1}^{m_1} \cdot {q_2}^{m_2} \cdot {q_3}^{m_3} \ldots {q_k}^{m_k}$$

then $2n$ *must divide the greatest common divisor (GCD) of* $(q_1 - 1, q_2 - 1, q_3 - 1, \ldots, q_k - 1)$.

Many researchers proposed various moduli that might satisfy the requirements, such as Mersenne [27] and Fermat [50] prime numbers. Here we define *NTT-friendly modulus* based on its abilities to perform the type of convolutions:

Definition 3.7: A ***PWC-NTT friendly*** *modulus* q *is defined if and only if an* n*-th root of unity,* ω, *exists in* $\mathbb{Z}_q$.

Definition 3.8: An ***NWC-NTT friendly*** *modulus* q *is defined if and only if* n*-th root of unity,* ω, *and* $2n$*-th root of unity,* ψ,*exists in* $\mathbb{Z}_q$.

In the schemes proposed for the NIST-PQC competition, the values of n and q are standardized. Table 1 summarizes the schemes and their NTT-friendliness.

TABLE 1. The values of n and q of standardized NIST-PQC scheme.

Scheme	n	q	PWC-NTT Friendly	NWC-NTT Friendly
Dilithium [10]	256	8380417	✔	✔
Falcon [12]	512	12289	✔	✔
	1024	12289	✔	✔
Kyber [15], [16]	256	3329	✔	✘

In the context of Post-Quantum Cryptography and Homomorphic Encryption, most of the time, the term "Number Theoretic Transform" and "Convolutions" refer to their negacyclic or negative-wrapped version. Therefore, for the rest of the report, we refer to all the terms "NTT", "INTT," and "convolutions" for their negative-wrapped version.

IV. FAST NTT

To reduce the complexity and fasten the process of the matrix multiplication needed for the NTT transformation, one can use "divide and conquer" techniques by utilizing the periodicity and symmetry property of ψ:

$$\text{periodicity:} \psi^{k+2n} = \psi^k \tag{19}$$

$$\text{symmetry:} \psi^{k+n} = -\psi^k \tag{20}$$

where k is a non-negative integer. The calculation of n point NTT and INTT can be divided into two $n/2$ points. However, the dividing techniques for NTT and INTT are slightly different.

A. COOLEY-TUKEY (CT) ALGORITHM FOR FAST-NTT

From equation (15), one can separate the summation into two parts based on the summation index parity:

$$\begin{aligned}\hat{\boldsymbol{a}}_j &= \sum_{i=0}^{n-1} \psi^{2ij+i} a_i \mod q \\ &= \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i} + \sum_{i=0}^{n/2-1} \psi^{4ij+2j+2i+1} a_{2i+1} \mod q \\ &= \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i} + \psi^{2j+1} \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i+1} \mod q\end{aligned} \tag{21}$$

Based on the ψ's symmetry properties:

$$\begin{aligned}\hat{\boldsymbol{a}}_{j+n/2} &= \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i} \\ &\quad - \psi^{2j+1} \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i+1} \mod q\end{aligned} \tag{22}$$

Let $A_j = \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i}$ and $B_j = \sum_{i=0}^{n/2-1} \psi^{4ij+2i} a_{2i+1}$, equations (21) and (22) become:

$$\begin{aligned}\hat{\boldsymbol{a}}_j &= A_j + \psi^{2j+1} B_j \mod q \\ \hat{\boldsymbol{a}}_{j+n/2} &= A_j - \psi^{2j+1} B_j \mod q\end{aligned} \tag{23}$$

Notice that A_j and B_j can be obtained as $n/2$ points NTT. If n is power-of-two, the process can be repeated for all the coefficients. Figure 4 shows the visualization of Equation (23), usually called *CT butterfly* as a reference to its proposer, Cooley and Tukey [23].

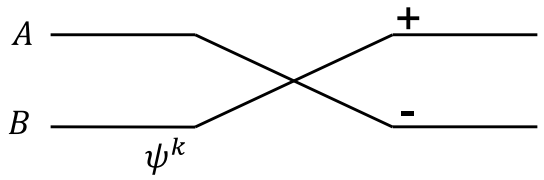


FIGURE 4. Cooley-Tukey (CT) butterfly unit for calculating NTT.

One can configure several butterfly units to calculate the entire n length of NTT.

Example 4.1: From Example 3.8, one can calculate the NTT by the matrix multiplication:

$$\hat{\boldsymbol{g}} = \begin{bmatrix} \psi^0 & \psi^1 & \psi^2 & \psi^3 \\ \psi^0 & \psi^3 & \psi^6 & \psi^9 \\ \psi^0 & \psi^5 & \psi^{10} & \psi^{15} \\ \psi^0 & \psi^7 & \psi^{14} & \psi^{21} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Based on the ψ periodicity:

$$\hat{\boldsymbol{g}} = \begin{bmatrix} \psi^0 & \psi^1 & \psi^2 & \psi^3 \\ \psi^0 & \psi^3 & \psi^6 & \psi^1 \\ \psi^0 & \psi^5 & \psi^2 & \psi^7 \\ \psi^0 & \psi^7 & \psi^6 & \psi^1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Based on the ψ symmetry:

$$\hat{\boldsymbol{g}} = \begin{bmatrix} \psi^0 & \psi^1 & \psi^2 & \psi^3 \\ \psi^0 & \psi^3 & -\psi^2 & \psi^1 \\ \psi^0 & -\psi^1 & \psi^2 & -\psi^3 \\ \psi^0 & -\psi^3 & -\psi^2 & \psi^1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Breaking down for each element:

$$\begin{aligned}\hat{g}_0 &= 1\psi^0 + 2\psi^1 + 3\psi^2 + 4\psi^3 \\ \hat{g}_1 &= 1\psi^0 + 2\psi^3 - 3\psi^2 + 4\psi^1 \\ \hat{g}_2 &= 1\psi^0 - 2\psi^1 + 3\psi^2 - 4\psi^3 \\ \hat{g}_3 &= 1\psi^0 - 2\psi^3 - 3\psi^2 + 4\psi^3\end{aligned}$$

Factoring:

$$\begin{aligned}\hat{g}_0 &= \psi^0(1 + 3\psi^2) + \psi^1(2 + 4\psi^2) \\ \hat{g}_1 &= \psi^0(1 - 3\psi^2) + \psi^3(2 + 4\psi^2) \\ \hat{g}_2 &= \psi^0(1 + 3\psi^2) - \psi^1(2 - 4\psi^2) \\ \hat{g}_3 &= \psi^0(1 - 3\psi^2) - \psi^3(2 - 4\psi^2)\end{aligned} \tag{24}$$

The idea is to calculate similar terms in the brackets once and then distribute the results instead of calculating them multiple times. Figure 5 shows the visualization of Equation 24.

The number of stages required is $\log_2(n)$. For our case here, as $n = 4$, two stages are required. For this example, the result of stage 1 is [2469, 5853, 5214, 1832]*, and stage 2 is*

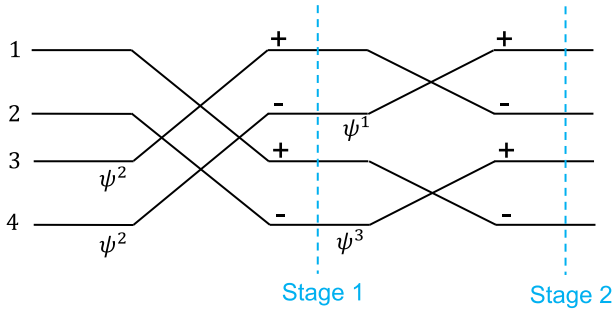


FIGURE 5. Cooley-Tukey butterflies for $n = 4$ and [1, 2, 3, 4] as its input.

[1467, 3471, 2807, 7621]. *By reordering the result of stage 2, we can get the correct NTT result:* [1467, 2807, 3471, 7621]

The order of the results of CT-Butterfly is called *bit-reversed order* (BO), while the correct order of the NTT is called *normal order* (NO). We will discuss the ordering in more detail in Subsection IV-C.

Example 4.2: Redoing Example 3.9, using the same butterfly configuration as Figure 5 with [5, 6, 7, 8] *as the input, the result of stage 1 is* [643, 4027, 7048, 3666]*, and stage 2 is* [2489, 6478, 7489, 6607]. *Reorder it to normal order for the NTT result:* [2489, 7489, 6478, 6607].

However, to calculate INTT, one will need another but similar "divide and conquer" approach.

B. GENTLEMAN-SANDE (GS) ALGORITHM FOR FAST-INTT

For the INTT, instead of dividing the summation by its index parity, it is separated by the lower and upper half of the summation. From equation (15) and ignoring n^{-1} term:

$$\begin{aligned}\boldsymbol{a}_i &= \sum_{j=0}^{n-1} \psi^{-(2i+1)j}\hat{a}_j \mod q \\ &= \left[\sum_{j=0}^{\frac{n}{2}-1} \psi^{-(2i+1)j}\hat{a}_j + \sum_{j=\frac{n}{2}}^{n-1} \psi^{-(2i+1)(j+\frac{n}{2})}\hat{a}_{(j+\frac{n}{2})}\right] \mod q \\ &= \psi^{-i}\left[\sum_{j=0}^{\frac{n}{2}-1} \psi^{-2ij}\hat{a}_j + \sum_{j=0}^{\frac{n}{2}-1} \psi^{-2i(j+\frac{n}{2})}\hat{a}_{(j+\frac{n}{2})}\right] \mod q\end{aligned}$$

Based on the periodicity and symmetry of ψ^{-1}, for the even term:

$$\boldsymbol{a}_{2i} = \psi^{-2i}\left[\sum_{j=0}^{\frac{n}{2}-1} \psi^{-4ij}\hat{a}_j + \sum_{j=0}^{\frac{n}{2}-1} \psi^{-4i(j+\frac{n}{2})}\hat{a}_{(j+\frac{n}{2})}\right] \mod q$$

$$\boldsymbol{a}_{2i} = \psi^{-2i}\sum_{j=0}^{\frac{n}{2}-1}\left[\hat{a}_j + \hat{a}_{(j+\frac{n}{2})}\right]\psi^{-4ij} \mod q \quad (25)$$

Doing the same derivation for the odd term:

$$\boldsymbol{a}_{2i+1} = \psi^{-2i}\sum_{j=0}^{\frac{n}{2}-1}\left[\hat{a}_j - \hat{a}_{(j+\frac{n}{2})}\right]\psi^{-4ij} \mod q \quad (26)$$

Let $A_i = \sum_{j=0}^{\frac{n}{2}-1} \hat{a}_j\psi^{-4ij}$ and $B_i = \sum_{j=0}^{\frac{n}{2}-1} \hat{a}_{j+\frac{n}{2}}\psi^{-4ij}$,

Equation (25) and (26) become:

$$\begin{aligned}\boldsymbol{a}_{2i} &= (A_i + B_i)\psi^{-2i} \mod q \\ \boldsymbol{a}_{2i+1} &= (A_i - B_i)\psi^{-2i} \mod q\end{aligned} \quad (27)$$

Notice that A_i and B_i can be obtained as $n/2$ points INTT. If n is power-of-two, the process can be repeated for all the coefficients. Figure 4 shows the visualization of Equation (27), usually called *GS butterfly* as a reference to its proposer, Gentleman and Sande [24].

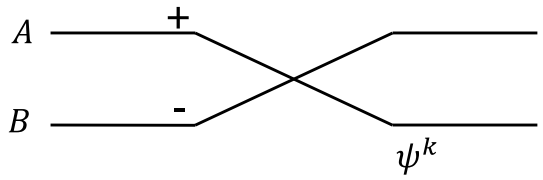


FIGURE 6. Gentleman-Sande (GS) butterfly unit for calculating INTT.

Because the separation is done differently, GS butterflies' input is usually in bit-reversed order (BO) and the output is in normal order (NO).

Example 4.3: Repeating example 3.10, let $NTT^{\psi}(\boldsymbol{g}) = \hat{\boldsymbol{g}} = [1467, 2807, 3471, 7621]$*, the INTT can be calculated by using matrix multiplication*:

$$\boldsymbol{g} = n^{-1}\begin{bmatrix}\psi^{-0} & \psi^{-0} & \psi^{-0} & \psi^{-0}\\ \psi^{-1} & \psi^{-3} & \psi^{-5} & \psi^{-7}\\ \psi^{-2} & \psi^{-6} & \psi^{-10} & \psi^{-14}\\ \psi^{-3} & \psi^{-9} & \psi^{-15} & \psi^{-21}\end{bmatrix}\begin{bmatrix}1467\\2807\\3471\\7621\end{bmatrix}$$

Based on ψ^{-1} *periodicity*:

$$\boldsymbol{g} = n^{-1}\begin{bmatrix}\psi^{-0} & \psi^{-0} & \psi^{-0} & \psi^{-0}\\ \psi^{-1} & \psi^{-3} & \psi^{-5} & \psi^{-7}\\ \psi^{-2} & \psi^{-6} & \psi^{-2} & \psi^{-6}\\ \psi^{-3} & \psi^{-1} & \psi^{-7} & \psi^{-5}\end{bmatrix}\begin{bmatrix}1467\\2807\\3471\\7621\end{bmatrix}$$

Based on ψ^{-1} *symmetry*:

$$\boldsymbol{g} = n^{-1}\begin{bmatrix}\psi^{-0} & \psi^{-0} & \psi^{-0} & \psi^{-0}\\ \psi^{-1} & \psi^{-3} & -\psi^{-1} & -\psi^{-3}\\ \psi^{-2} & -\psi^{-2} & \psi^{-2} & -\psi^{-2}\\ \psi^{-3} & \psi^{-1} & -\psi^{-3} & -\psi^{-1}\end{bmatrix}\begin{bmatrix}1467\\2807\\3471\\7621\end{bmatrix}$$

Breaking down for each element:

$$\begin{aligned}g_0 &= [1467\psi^{-0} + 2807\psi^{-0} + 3471\psi^{-0} + 7621\psi^{-0}]n^{-1}\\ g_1 &= [1467\psi^{-1} + 2807\psi^{-3} - 3471\psi^{-1} - 7621\psi^{-3}]n^{-1}\\ g_2 &= [1467\psi^{-2} - 2807\psi^{-2} + 3471\psi^{-2} - 7621\psi^{-2}]n^{-1}\\ g_3 &= [1467\psi^{-3} + 2807\psi^{-1} - 3471\psi^{-3} - 7621\psi^{-1}]n^{-1}\end{aligned}$$

Factoring:

$$\begin{aligned} g_0 &= [(1467 + 3471)\psi^{-0} + (2807 + 7621)\psi^{-0}]\psi^{-0}n^{-1} \\ g_1 &= [(1467 - 3471)\psi^{-1} + (2807 - 7621)\psi^{-3}]\psi^{-0}n^{-1} \\ g_2 &= [(1467 + 3471)\psi^{-0} - (2807 + 7621)\psi^{-0}]\psi^{-2}n^{-1} \\ g_3 &= [(1467 - 3471)\psi^{-1} - (2807 - 7621)\psi^{-3}]\psi^{-2}n^{-1} \end{aligned} \tag{28}$$

Similar to NTT, the idea is to calculate the similar terms in the brackets once, then distribute the results instead of calculating them multiple times. By first reordering the input, we can visualize Equation 28 as shown in Figure 7.

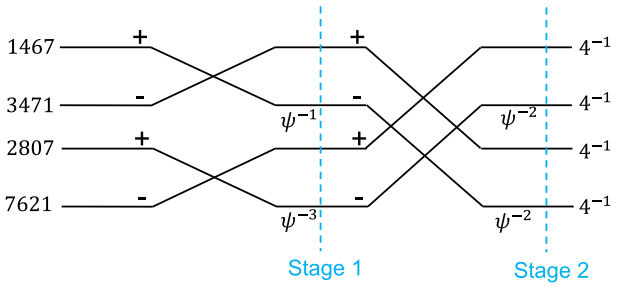


FIGURE 7. Gentleman-Sande butterflies for $n = 4$ and [1467, 2807, 3471, 7621] reordered as bit-reversed order as its input.

The result of stage 1 is [4938, 4025, 2747, 3664]*, and stage 2 is* [4, 8, 12, 16]*. After scaling with a* $4^{-1} = 5761$ *factor, we can get the INTT result of* [1, 2, 3, 4].

Example 4.4: Redoing Example 3.11, using the same butterfly configuration as Figure 7, reordering the input from normal order [2489, 7489, 6478, 6607] *to bit-reversed order* [2489, 6478, 7489, 6607]*. The result of stage 1 is* [1286, 373, 6415, 7332]*, the result of stage 2 is* [20, 14, 28, 32]*, and the INTT result after scaling is* [5, 6, 7, 8].

For polynomial multiplication, one can use CT butterflies to transform both inputs to the NTT domain, then use element-wise multiplication for the NTT outputs. The result is then transformed back using GS butterflies to perform INTT. As the butterflies reduce the mathematical operation in a quasilinear scale, the complexity of the polynomial multiplication is reduced from $O(n^2)$ to $O(n \log n)$. The larger the polynomial degree, the larger the speed and cost gain [51].

Example 4.5: From example 4.1, we get that the NTT transformation of [1, 2, 3, 4] *in bit-reversed order is* [1467, 3471, 2807, 7621]*. From example 4.2, we get the NTT transformation of* [5, 6, 7, 8] *is* [2489, 6478, 7489, 6607] *in bit-reversed order. Using element-wise multiplication for those two results, we get* [2888, 2851, 6407, 2992] *in bit-reversed order. Transforming back the results using GS-butterfly, we will get* [7625, 7645, 2, 60] *or* [−56, −36, 2, 60] *when written using negative numbers. Which is the same result as Example 3.12.*

TABLE 2. Normal and bit-reversed order for $n = 4$.

Index	Index in binary	Bit-reversal	Decimal of bit-reversal
0	00	00	0
1	01	10	2
2	10	01	1
3	11	11	3

TABLE 3. Normal and bit-reversed order for $n = 8$.

Index	Index in binary	Bit-reversal	Decimal of bit-reversal
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

C. NORMAL ORDER AND BIT-REVERSED ORDER

As encountered in Subsection IV-A and IV-B, typically, the input of CT Butterfly is in Normal Order (NO), and the output is in Bit-reversed Order (BO). Conversely, the input of GS Butterfly is in BO, and the output is in NO. This section clarifies the formal definition of Normal and Bit-reversed Order and provides examples for $n = 4$ and $n = 8$.

Definition 4.1: Let n be a power of two, and b is a non-negative integer with $b < n$*. The* ***bit-reversal*** *of b is defined as*:

$$\begin{aligned} &brv_n(b_{\log n-1}2^{\log n-1} + \cdots + b_1 2 + b_0) \\ &\quad = b_0 2^{\log n-1} + \cdots + b_{\log n-2}2 + b_{\log n-1} \end{aligned}$$

where b_i *is the i-th bit of the binary expansion of b [33].*

Example 4.6: Consider $n = 4$*, the index of the array in the normal order is* [0, 1, 2, 3]*. Table 2 shows the index binary representation in* $\log_2 n = 2$ *bit, their bit-reversal in binary, and their decimal representation.*

From the table, we know that the index of the normal order is [0, 1, 2, 3] *and the index of the bit-reversed order is* [0, 2, 1, 3]

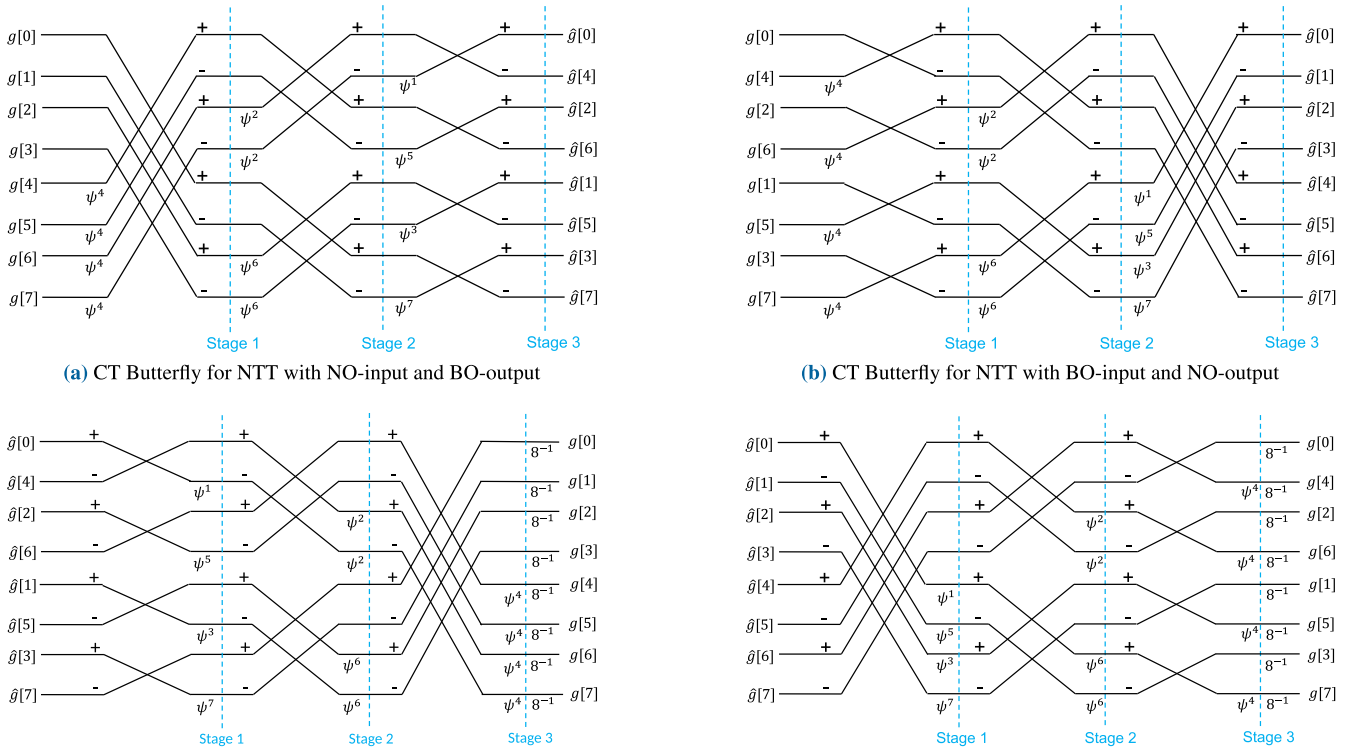
Example 4.7: Similarly, when considering $n = 8$*, we can construct a similar table with* $\log_2 n = 3$ *as the length of binary representation.*

We will get the NO index is [0, 1, 2, 3, 4, 5, 6, 7]*, and the BO index is* [0, 4, 2, 6, 1, 5, 3, 7].

Example 4.8: Redoing the previous examples for $n = 16$ *and* 4 *as the length of binary representation.*

Therefore the NO index is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 12, 13, 14, 15] *and the BO index is* [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15].

Typical NTT-CT Butterfly configuration has NO-input and BO-output, while INTT-GS configuration usually has BO-input and NO-output. However, one can reconfigure the CT butterfly to have BO-input & NO-output and GS butterfly



(a) CT Butterfly for NTT with NO-input and BO-output

(b) CT Butterfly for NTT with BO-input and NO-output

(c) GS Butterfly for INTT with BO-input and NO-output

(d) GS Butterfly for INTT with NO-input and BO-output

FIGURE 8. All possible CT and GS butterfly configurations for $n = 8$.

TABLE 4. Normal and bit-reversed order for $n = 16$.

Index	Index in binary	Bit-reversal	Decimal of bit-reversal
0	0000	0000	0
1	0001	1000	8
2	0010	0100	4
3	0011	1100	12
4	0100	0010	2
5	0101	1010	10
6	0110	0110	6
7	0111	1110	14
8	1000	0001	1
9	1001	1001	9
10	1010	0101	5
11	1011	1101	13
12	1100	0011	3
13	1101	1011	11
14	1110	0111	7
15	1111	1111	15

to have NO-input & BO-output. Figure 8 shows all possible configurations for NTT CT and INTT GS Butterfly for $n = 8$.

Using normal order as NTT input is called *decimation in time*, while bit-reversed order input is called *decimation in frequency* [52]. Another thing to notice is that the power of ψ follows the bit-reversed order index. The set of all the exponentiation of ψ is called *twiddle factors*.

D. MODULAR ARITHMETIC UNITS

One of the challenges of NTT-based multiplication is that addition and multiplication have to be done in $\mathbb{Z}_q$. All the CT and GS butterflies operators require modular arithmetic, a non-standard feature for most implementation platforms.

1) MODULAR ADDER

To calculate modular addition, $(A+B) \bmod q$, we can simply use a piece-wise function:

$$(A+B) \bmod q = \begin{cases} A+B & A+B < q \\ A+B-q & A+B \geq q \end{cases} \tag{29}$$

Equation (29) is relatively easy to implement using a set of adders, subtractors, and multiplexers.

While modular adder is simple and easy to implement, modular multipliers are trickier. The standard algorithm for modular multiplication uses trial division, which is inefficient, not scalable, and difficult to implement in hardware architecture. The most popular workaround for implementing Barrett or Montgomery modular multiplication algorithm.

2) MODULAR REDUCTION: BARRETT METHOD

The main idea behind Barrett reduction is to approximate the division by the modulus using pre-computed values, which allows for faster modular multiplication [53], [54]. Algorithm 1 shows how to multiply two integers modulo q using Barrett reduction. As the value of q is usually fixed,

Algorithm 1 Modular Multiplication by Barrett Reduction

Input: $a, b, q \in \mathbb{Z}$
Output: $a \times b \bmod q$
// Pre-computation
$k = \lceil \log_2 q \rceil$ // number of bits in q
$r = 2^k$
$\mu = \lfloor \frac{r^2}{q} \rfloor$
// Multiplication
$z = a \times b$
// Barrett Reduction
$m_1 = \lfloor \frac{z}{r} \rfloor$
$m_2 = m_1 \times \mu$
$m_3 = \lfloor \frac{m_2}{r} \rfloor$
$t = z - m_3 \times q$
if $t \geq q$ **then**
  return $t - q$
else
  return t
end if

Algorithm 2 Modular Multiplication by Montgomery Reduction

Input: $a, b, q \in \mathbb{Z}$
Output: $a \times b \bmod q$
// Pre-computation
$k = \lceil \log_2 q \rceil$ // number of bits in q
$r = 2^k$
$r_{\text{inv}} = r^{-1} \bmod q$
$q' = -q^{-1} \bmod r$
// Convert to Montgomery representation
$a_m = a \times r \bmod q$
$b_m = b \times r \bmod q$
// Multiplication in Montgomery representation
$t = a_m \times b_m$
$u = t \times q' \bmod r$
$c_m = (t + u \times q)/r$
// Convert back to the standard representation
$c = c_m \times r_{\text{inv}} \bmod q$
return c

we can pre-compute the value of k, r, and μ while designing the unit. The floor function of division by r, a power-of-two integer, can be replaced by the right shift function that is easy, cheap, and efficient to implement in hardware.

This method is suitable for modular multiplication between unrelated numbers [55]. In the CT and GS Butterfly, this type of multiplication is used to multiply the polynomial coefficients with various twiddle factors. It is also used in the element-wise multiplication between two NTT vectors.

Example 4.9: To calculate $1467 \times 2489 \bmod 7681$ *(used in Example 3.12) using Barrett reduction, one will get* $a = 1467$, $b = 2489$, *and* $q = 7681$.

$$
\begin{aligned}
k &= \lceil \log_2 q \rceil = \lceil \log_2 7681 \rceil = 13 \\
r &= 2^k = 2^{13} = 8192 \\
\mu &= \lfloor \frac{r^2}{q} \rfloor = \lfloor \frac{8192^2}{7681} \rfloor = 8736 \\
z &= a \times b = 1467 \times 2489 = 3651363 \\
m_1 &= \lfloor \frac{z}{r} \rfloor = \lfloor \frac{3651363}{8192} \rfloor = 445 \\
m_2 &= m_1 \times \mu = 445 \times 8736 = 3887520 \\
m_3 &= \lfloor \frac{m_2}{r} \rfloor = \lfloor \frac{3887520}{8192} \rfloor = 474 \\
t &= z - m_3 \times q = 3651363 - 474 \times 7681 = 10569
\end{aligned}
$$

As $t \geq 7681$, *the result is* $t - q = 10569 - 7681 = 2888$.

3) MODULAR REDUCTION: MONTGOMERY METHOD

Another alternative is to perform modular multiplication by Montgomery reduction [56], [57], which is shown in Algorithm 2. The main idea is that it avoids direct divisions by the modulus by transforming the number to the Montgomery representation.

The main drawback of Montgomery reduction is the requirement to transform the numbers into Montgomery representation in $\mathbb{Z}_q$ in contrast to Barrett reduction, in which all the calculation is done in $\mathbb{Z}$. However, this drawback can also be advantageous when calculating the same multiplication multiple times. Hence, this method is suitable for modular exponentiation [55]. In the case of NTT, it is useful to calculate the exponentiation of ψ used in CT and GS butterflies as twiddle factors.

Example 4.10: To calculate $1467 \times 2489 \bmod 7681$ *(used in Example 3.12) using Montgomery reduction, one will get* $a = 1467$, $b = 2489$, *and* $q = 7681$.

Pre-computation:

$$
\begin{aligned}
k &= \lceil \log_2 q \rceil = \lceil \log_2 7681 \rceil = 13 \\
r &= 2^k = 2^{13} = 8192 \\
r_{inv} &= r^{-1} \bmod q = 8192^{-1} \bmod 7681 = 7200 \\
q' &= -q^{-1} \bmod r = -(7681^{-1}) \bmod 8192 = 7679
\end{aligned}
$$

Convert a and b to Montgomery representation:

$$
\begin{aligned}
a_m &= a \times r \bmod q = 1467 \times 8192 \bmod 7681 = 4580 \\
b_m &= b \times r \bmod q = 2489 \times 8192 \bmod 7681 = 4514
\end{aligned}
$$

Multiplication in Montgomery representation:

$$
\begin{aligned}
t &= a_m \times b_m = 4580 \times 4514 = 20674120 \\
u &= t \times q' \bmod r = 20674120 \times 7679 \bmod 8192 = 6584
\end{aligned}
$$

Result in Montgomery representation:

$$
\begin{aligned}
c_m &= (t + u \times q)/r \\
&= (20674120 + 6584 \times 7681)/8192 \\
&= 8697
\end{aligned}
$$

Transform back to standard representation:

$$\begin{aligned} c &= c_m \times r_{inv} \bmod q \\ &= 8697 \times 7200 \bmod 7681 \\ &= 2888 \end{aligned}$$

Transforming to and from Montgomery representation is an expensive operation, which is usually done iteratively by subtracting q multiple times. One needs to minimize the number of transformations to use Montgomery modular reduction efficiently.

Many researchers perform various workarounds and optimizations for NTT/INTT implementation using previously discussed concepts in various Post-Quantum Cryptography applications, which we will discuss in the following chapter.

V. NTT IN POST QUANTUM CRYPTOGRAPHY SCHEME

All the NIST-PQC competition winners: Dilithium [10], Falcon [12], and Kyber [15], [16] include NTT/INTT in their specifications for modular polynomial multiplication. In this section, we surveyed the implementation of NTT/INTT for each scheme in various platforms based on their novelty claims, algorithms, and implementation strategies. We also present common optimizations implemented by various researchers.

A. DILITHIUM, KYBER, AND FALCON OVERVIEW

Dilithium [10] is one of the standardized algorithms in the NIST Post-Quantum Cryptography (PQC) competition. It is a signature scheme based on the problem of finding short lattice vectors, which is believed to be hard even for quantum computers. The Dilithium algorithm is designed to provide strong security while remaining efficient enough for practical use in digital signature applications.

Kyber [15], [16] is a key encapsulation mechanism (KEM) part of the NIST Post-Quantum Cryptography (PQC) project. Kyber is one of the proposed algorithms in the NIST-PQC competition. It is a lattice-based cryptosystem that relies on the hardness of the Learning With Errors (LWE) problem and its variants, which are believed to resist quantum attacks.

Falcon [12] is one of the candidate algorithms for digital signature schemes In NIST-PQC (National Institute of Standards and Technology Post-Quantum Cryptography). Falcon is a family of lattice-based signature schemes designed to be secure against attacks by quantum computers. Falcon uses a variation of the Ring Learning With Errors (RLWE) problem, which is believed to be resistant to attacks by both classical and quantum computers. The security of Falcon relies on the hardness of the underlying mathematical problem of finding the shortest vector in a lattice. Falcon provides efficient signature generation and verification, making it a practical option for real-world applications. It is also designed to resist side-channel attacks, which exploit weaknesses in the physical implementation of a cryptographic system.

TABLE 5. The values of n, q, ω, and ψ of standardized NIST-PQC scheme. Note that only Dilithium specifies the actual value of ψ, others do not.

Scheme	n	q	ω	ψ
Dilithium [10]	256	8380417	3073009	1753
Falcon [12]	512	12289	3	1321
	1024	12289	49	12282
Kyber [15], [16]	256	3329	17	-

This report highlights the NTT/INTT specifications and the Dilithium, Kyber, and Falcon scheme implementations. Optimizations and various implementations outside the NTT/INTT in the scheme are out of the scope of our work.

B. NTT IN FINALIZED PQC SCHEMES

NTT is a part of Dilithium specification, with the parameters set as polynomials of degree $n = 256$ and the modulus $q = 2^{23} - 2^{13} + 1 = 8380417$ is used in the extended cyclotomic ring $\mathbb{Z}_q[x]/(x^{256} + 1)$. Notice that the chosen q is NWC-NTT friendly prime where ψ exists. Dilithium also specifies the chosen $2n$-th root of unity, $\psi = 1753$. These parameters were chosen based on a trade-off between security and efficiency [10].

NTT is also a part of Falcon and Kyber specifications. Falcon [12] specifies that $n = 512$ or $n = 1024$ depends on the desired security level and the modulus q is chosen to be 12289, which is an NWC-NTT friendly modulus for both n. Kyber [16] also specifies $n = 256$ and $q = 3329$ in its finalized version. Table 5 shows the NTT parameters summary for Dilithium, Falcon, and Kyber.

As we can see, the NTT specification in Kyber is unique because the chosen modulus in the final version, $q = 3329$, is not an NTT-NWC friendly modulus, which requires a special trick called *truncated NTT* to calculate its negative-wrapped convolution. Truncated NTT requires the calculations of NTT divided into two parts, as for Kyber $n = 256$, it requires two NTT calculations with $n = 128$ by dividing odd and even parts [58]. Notice that when $n = 128$ and $q = 3329$, it is an NWC-NTT friendly modulus with one of the $\psi = 892$. In the following toy example for truncated NTT, we can calculate NTT/INTT with $n = 8$ by breaking it down into two NTT/INTT calculations with $n = 4$.

Example 5.1: Let $\boldsymbol{A} = [0, 1, 2, 3, 4, 5, 6, 7]$ *and* $\boldsymbol{B} = [8, 9, 10, 11, 12, 13, 14, 15]$ *in the ring* $\mathbb{Z}_Q$ *with* $Q = 7681$. *We need to find the negacyclic convolution of* $\boldsymbol{A}$ *and* $\boldsymbol{B}$.

Calculating the results using previously explained methods in normal order: *Using* $\psi = 7154$, *we can get*:

$$NTT^{\psi}(\boldsymbol{A}) = [0, 7154, 2426, 2497, 1830, 4245, 3812, 4081]$$
$$NTT^{\psi}(\boldsymbol{B}) = [8, 2938, 4449, 4035, 5490, 3356, 3774, 1064]$$

Element-wise multiplication between the two yields:

$$\begin{aligned} &NTT^{\psi}(\boldsymbol{A}) \circ NTT^{\psi}(\boldsymbol{B}) \\ &\quad = [3213, 7391, 1790, 5474, 5572, 2527, 2633, 7341] \end{aligned}$$

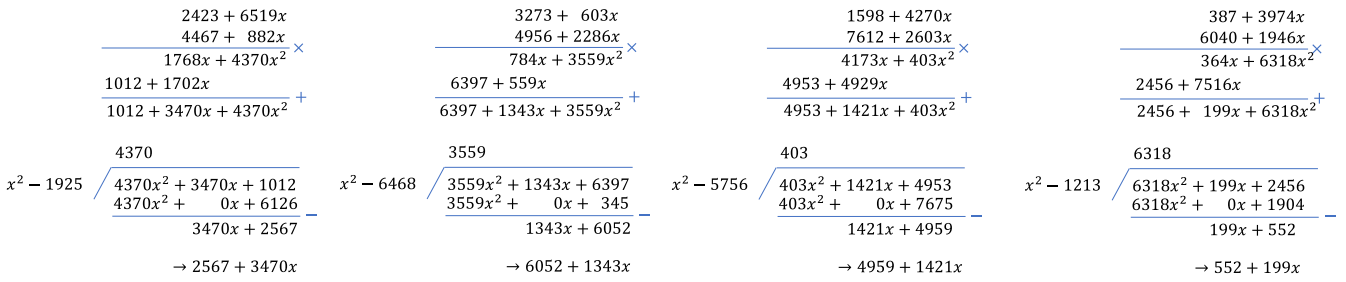


FIGURE 9. Schoolbook multiplication instead of element-wise for Example 5.1.

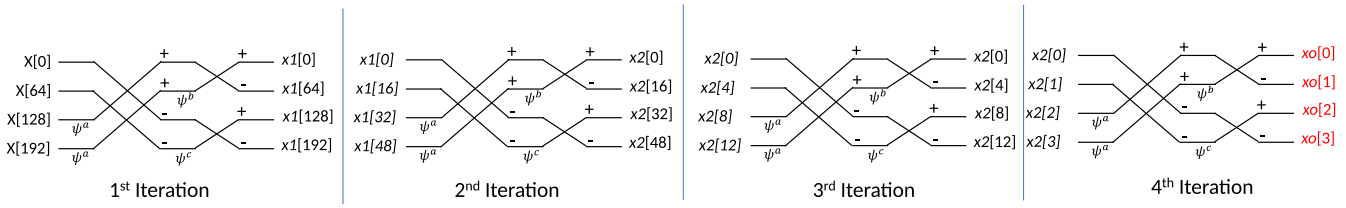


FIGURE 10. Calculating 256-element NTT Transformation using 4-element CT butterfly iteratively.

Taking INTT from the results yields in the negacyclic convolution between $\boldsymbol{A}$ *and* $\boldsymbol{B}$

$$INTT^{\psi^{-1}}(NTT^{\psi}(\boldsymbol{A}) \circ NTT^{\psi}(\boldsymbol{B})) = [7373, 7369, 7391, 7441, 7521, 7633, 98, 280]$$

Therefore the negacyclic convolution between them is [7373, 7369, 7391, 7441, 7521, 7633, 98, 280].

Calculating the results by truncated NTT: *Group* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *into two*:

$$\boldsymbol{A} = [(0, 1), (2, 3), (4, 5), (6, 7)]$$
$$\boldsymbol{B} = [(8, 9), (10, 11), (12, 13), (14, 15)]$$

Then, separate it into odd and even parts:

$$\boldsymbol{A}_{even} = [0, 2, 4, 6]$$
$$\boldsymbol{A}_{odd} = [1, 3, 5, 7]$$
$$\boldsymbol{B}_{even} = [8, 10, 12, 14]$$
$$\boldsymbol{B}_{odd} = [9, 11, 13, 15]$$

Transform them into NTT with $n = 4, q = 7681$, *and* $\psi = 1925$*:*

$$NTT^{\psi}(\boldsymbol{A}_{even}) = [2423, 3273, 1598, 387]$$
$$NTT^{\psi}(\boldsymbol{A}_{odd}) = [6519, 603, 4270, 3974]$$
$$NTT^{\psi}(\boldsymbol{B}_{even}) = [4467, 4956, 7612, 6040]$$
$$NTT^{\psi}(\boldsymbol{B}_{odd}) = [882, 2286, 2603, 1946]$$

Note that we present the result here in normal order. We can group them again:

$$NTT^{\psi}(\boldsymbol{A}) = [(2423, 6519), (3273, 603), (1598, 4270), (387, 3974)]$$
$$NTT^{\psi}(\boldsymbol{B}) = [(4467, 882), (4956, 2286), (7612, 2603), (6040, 1946)]$$

TABLE 6. Values of ψ^{2i+1}, which is important to determine the modulus of the schoolbooks multiplication.

Index, i	0	1	2	3
$2i+1$	1	3	5	7
ψ^{2i+1}	1925	6468	5756	1213

In the usual NTT/INTT scheme, we multiply element-wise in this step. However, in this scheme, we do a schoolbook multiplication between each grouped element treated as polynomial coefficients [59]. The schoolbook multiplication is performed modulo $X^2 - \psi^{2i+1}$*, where i is the index of grouped elements. Figure 9 illustrates how our example case is performed, while Table 10 shows the* ψ^{2i+1} *values.*

From the calculations, we obtain the following:

$$\hat{\boldsymbol{C}} = NTT^{\psi}(\boldsymbol{A}) \circ NTT^{\psi}(\boldsymbol{B}) = [(2567, 3470), (6052, 1343), (4959, 1421), (552, 199)]$$

Separating odd and even values of $\hat{\boldsymbol{C}}$*:*

$$\hat{\boldsymbol{C}}_{even} = [2567, 6052, 4959, 552]$$
$$\hat{\boldsymbol{C}}_{odd} = [3470, 1343, 1421, 199]$$

Transform them back using INTT:

$$INTT^{\psi^{-1}}(\hat{\boldsymbol{C}}_{even}) = [7373, 7391, 7521, 98]$$
$$INTT^{\psi^{-1}}(\hat{\boldsymbol{C}}_{odd}) = [7369, 7441, 7633, 280]$$

Grouping them:

$$\boldsymbol{C} = [(7373, 7369), (7391, 7441), (7521, 7633), (98, 280)]$$

Which is the same result as regular NTT/INTT negative wrapped convolution performed previously.

C. IMPLEMENTATION OPTIMIZATIONS

As the value of n and q is relatively small in the finalized PQC schemes, one can straightforwardly implement CT and GS butterflies for NTT/INTT with 256 points. Nevertheless, various optimizations on the implementation side exist. This section discusses what other researchers do to achieve their desired goals.

1) ITERATIVE CT/GS BUTTERFLIES

One may look to optimize the NTT/INTT for a smaller area in the hardware implementation. This goal can be achieved using an iterative approach, which involves performing the butterflies repeatedly with a trade-off for longer clock cycles.

This means one can break it into smaller units instead of doing a single 256-element CT/GS butterfly. For instance, since 256 can be expressed as $16^2 = 4^4 = 2^8$, one can do a 16-element CT/GS butterfly twice (called *radix-16*), a 4-element four times (called *radix-4*), or a 2-element eight times (called *radix-2*). This allows for a reduction in the area required to implement the butterflies while still maintaining the same transformation result.

However, the iterative method has another trade-off: the need to generate a different set of twiddle factors for each clock cycle. This means the twiddle factors must be recalculated for each iteration, adding to the processing time. This trade-off is necessary to ensure that the correct results are obtained at each stage of the iterative transformation.

Figure 10 illustrates how a 4-element CT butterfly can be used to calculate 256-element NTT transformation. For each clock cycle, the twiddle factors: ψ^a, ψ^b, and ψ^c are changed according to the input elements. The order of the four 4-elements inputted must also be managed carefully to ensure the correct results. This scheme can also be applied similarly to GS butterflies.

This optimization is common among researchers and used in almost all papers we surveyed due to its effectiveness in saving area and design flexibility.

2) BITWISE OPERATIONS FOR MODULAR REDUCTION

Another common optimization among researchers in this scheme is utilizing the fact that the modulus q is already set and fixed for each scheme. This fact can be exploited to design a custom modular reduction module specific to the targeted modulus by only using hardware-friendly operations such as add and shift.

For example, in Dilithium, the value of the modulus is fixed at $q = 8380417$. This number can also be written as $2^{23} - 2^{13} + 1$, then one can derive that $2^{23} \equiv 2^{13} - 1 \bmod q$. Similar derivation can also be done for Falcon and Kyber moduli. Instead of using Barrett or Montgomery reduction, a series of bit-wise operations based on this fact are done to perform modular multiplication.

The idea of designing a custom bit-wise operations modular reduction module is attributed to [60] by many researchers we surveyed.

3) HARDWARE SPECIFIC FEATURES UTILIZATION

Another common optimization among researchers is using more efficient and ready-to-use platforms in their targeted devices. Some platforms provide useful features, such as ready-to-use multipliers in the DSP module of FPGA, vector arithmetic, large register sets, specialized instruction sets, and parallelization. However, this optimization is very specific to the targeted device, and not all researchers can access the specialized hardware.

D. IMPLEMENTATIONS COMPARISON

To close this section, Table 7 shows the summary of the implementation comparisons between various researchers based on their novelty claim on NTT/INTT implementations algorithm, target device or hardware, their presented NTT/INTT implementations, and how they implement modular reduction algorithms.

VI. NTT IN HOMOMORPHIC ENCRYPTION ARCHITECTURES

Another use case of NTT is its application in Homomorphic Encryption (HE). This section explores the NTT implementations of HE schemes in various platforms, such as CPUs, GPUs, and FPGAs. NTT is one of the key components in Homomorphic Encryption (HE) algorithms that rely on polynomial arithmetic (addition and multiplication). NTT is an integer version of FFT, which is well-known to perform $O(n \log n)$ time complexity. Suppose we have two n-degrees polynomial numbers, A and B. Theoretically, it can be multiplied in time $O(n \log n)$ [118]. Moreover, they showed that polynomials degree less than $n \in \mathbb{R}_q$ with q elements cost $O(n \log q \log(n \log q))$. One big question is how to implement the $O(n \log n)$ complexity in the platforms such as GPUs and FPGAs. This section will answer it.

To have a comprehensive understanding of the architectures, first, we explain how the CPU, GPU, and FPGA work. Figures 11 and 12 show the CPU, GPU, and FPGA architecture, respectively. CPU is built on several modules such as registers, arithmetic logic unit (ALU), control unit, and instruction memory. The left side of Figure 11 shows a multi-core CPU that supports the execution of multiple instructions simultaneously. Typical multi-core CPUs are also supported by caches that temporarily hold the data and instructions to speed up the data movement rather than accessing the DRAM. However, the CPUs are specialized for serial processing with higher clock frequencies that may not be suitable for the data format of polynomial operations in homomorphic encryption.

GPU, or graphics processing unit, is a processor for handling complex mathematical and graphical computations. A GPU contains much more cores than the CPU *e.g.,* hundreds, thousands, even hundred thousands or million cores that can work simultaneously in parallel. The right side of Figure 11 shows the typical GPU architecture that consists of streaming multiprocessors (SM) containing the cores.

TABLE 7. Summary of Dilithium, Kyber, and Falcon's NTT hardware implementations.

Ref.	Implemented PQC scheme	Novelty claim on NTT/INTT implementations	Target device	NTT/INTT algorithms	Modular reduction
[61]	Dilithium and Kyber	Unified polynomial multiplier for Dilithium and Kyber in one system to minimize latency and implementation area.	ASIC: Cadence Genus, FPGA: Xilinx Zynq Ultrascale+ZCU102	Modified iterative CT and GS by unifying butterfly unit of Dilithium (1 operation) and Kyber (2 operations).	Bitwise add-shift operations based on [60].
[62]	Dilithium	Applying parallelization to the NTT-based polynomial multiplication utilizing the ARMv8 architectural features: large register sets and NEON engine.	NVIDIA Jetson AGX Xavier with 8-core ARM v8.2 64-bit CPU.	Iterative CT and GS with memory management optimization, such as merging and register holding.	Parallelized Montgomery reduction [56].
[63]	Dilithium	Memory conflict management allows efficient memory access and avoids stalls while pipelining, but still limits logic and area consumption.	FPGA: Xilinx Artix-7 and Virtex Ultrascale	Iterative CT and GS with memory management technique to allow parallel in place NTT.	Barrett reduction [53] implemented using hardware-friendly operation: add and shift.
[64]	Dilithium and Kyber.	Implementing Radix-4 based NTT/INTT with a pipelined and parallel data path.	FPGA: Xilinx Zynq-7000	Modified CT and GS butterfly to have a radix-4 instead of radix-2: a butterfly structure takes 4 elements at a time. The radix-4 implementation is based on [65].	Bitwise add-shift operations based on [60], [66].
[59]	Dilithium and Kyber	Utilizing floating-point registers of Cortex-M4 to reduce memory access time, using Fermat Number Transform (a special case of NTT with Fermat prime) for Dilithium.	STM32F4 Discovery board with ARM Cortex-M4	Both NTT and INTT calculations use CT butterflies instead of the usual CT-GS pairs.	Modified Barrett reduction [53] to utilize the hardware features of Cortex-M4.
[67]	Dilithium	Combined hybrid system to calculate both NTT and INTT to maintain speed-area trade-off	FPGA: Xilinx Zynq-7000 XC7Z020-1	Combining CT and GS butterflies in one module by designing specialized control logic and butterfly computation unit.	Custom bitwise add-shift operations.
[68]	Dilithium	Small area NTT/INTT implementation using only a single dual-butterfly unit attached to two 64×256 dual-port memories.	FPGA: Xilinx Artix-7, Zynq-7000, and Ultrascale+	Iterative CT and GS using butterfly units that calculate 128 operations at a time.	Bitwise add-shift operations based on [60].
[69]	Dilithium	Parallelization and optimization NTT/INTT for low-power Internet-of-Things (IoT) applications using the ARMv80A Neon SIMD Instructions.	Raspberry Pi 4B (RPi 4 Model B) with ARMv8-A instruction sets, Cortex-A72 (1.8 GHz) CPU, and 4GB RAM	Modifying CT/GS algorithm to fully utilize the fact that the operands are much smaller than the modulus, q.	Not explained in the reference.
[70]	Dilithium	Adopting an FFT technique called the Radix-2 Multipath Delay Commutator (R2MDC) [71] that requires fewer memory access into NTT.	FPGA: Xilinx Artix-7	CT/GS butterflies optimized with R2MDC in one module.	Bitwise operations using Canonical Signed Digit (CSD) representation [72].
[73]	Dilithium and Kyber	Optimization of NTT/INTT for an optimal Look-Up Table (LUT) and Flip-Flop (FF) utilization by efficient use of special-purpose Digital Signal Processors (DSPs)	FPGA: Xilinx Artix-7	Modified CT and GS butterflies based on [66]. For INTT, the scaling factor n is incorporated into the twiddle factors.	Bitwise add-shift operations based on [66]
[74]	Dilithium	Compact hardware architecture that can complete NTT/INTT in a short clock period.	FPGA: Altera DE2-115 and Xilinx Zynq-7020	Iterative CT for NTT and GS for INTT, with pipeline and customized twiddle factor generators to allow more flexible scheduling.	Karatsuba multiplication [18] with Montgomery reduction [56].
[75]	Dilithium and Falcon	Design an architecture that can be efficiently implemented using High-Level Synthesis (HLS) by memory management.	FPGA: Xilinx Zynq UltraScale+	Pipelined iterative CT and GS, with 2×2 butterfly unit, processing 4 coefficients simultaneously.	Bitwise add-shift operations, named K-RED based on [76].
[77]	Dilithium	Processing two layers of NTT/INTT at once by implementing 2×2 butterfly units combined with coefficient rearrangement to reduce stall while pipelining.	FPGA: Xilinx Virtex Ultrascale+, Artix-7, and Kintex-7	Pipelined iterative CT and GS, with 2×2 butterfly unit based on [75], processing 4 coefficients at a time.	Barrett reduction [53] with only add-shift operations.
[78]	Dilithium	Naive CT and GS NTT/INTT implementation without their iterative version is possible on the targeted device.	NXP FRDM-K64F Development Platform	Naive CT and GS NTT and INTT pairs.	Not explained in the article.

TABLE 7. *(Continued.)* **Summary of Dilithium, Kyber, and Falcon's NTT hardware implementations.**

Ref.	Implemented PQC scheme	Novelty claim on NTT/INTT implementations	Target device	NTT/INTT algorithms	Modular reduction
[79]	Dilithium	Designing and optimizing essential NTT/INTT functions in VHDL for hardware environments.	FPGA: Xilinx Virtex-7 Ultrascale+	Iterative CT and GS NTT/INTT using 2×2 butterfly units to calculate two iterations simultaneously.	Montgomery reduction [56].
[80]	Dilithium	Implementation of NTT/INTT using Neon instruction set and improving Barrett and Montgomery reduction, making it suitable for the target device.	Raspberry Pi 4 Model B with Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8)	Iterative CT and GS specifically designed to utilize the ARMv8 instruction set.	Combination of Barrett [53] and Montgomery [56] reduction, specifically designed to utilize Neon SIMD instructions from ARMv8.
[81]	Dilithium	Unifying butterfly unit module, making it able to function as CT butterfly, GS butterfly, and element-wise multiplier to reduce the area consumption.	FPGA: Xilinx Kintex-7 Sakura-X	Iterative CT and GS algorithm with controller logic to select the function of butterfly unit.	Customized bitwise add-shift operations.
[82]	Dilithium	Precomputing twiddle factors in Montgomery domain, the use of signed representation to prevent integer overflow.	STM32F4 Discovery board with ARM Cortex-M4	Iterative CT and GS, which the butterflies implement using the ARM Cortex-M4's instruction set.	Montgomery reduction designed specifically for utilizing the ARM Cortex-M4 instruction set.
[83]	Kyber	Performs truncated NTT/INTT tricks [33] with Karatsuba multiplication [18] to obtain the resulting polynomial.	FPGA: Xilinx Artix-7 and Zynq UltraScale+	Iterative CT and GS with customized 2×2 butterflies that can function as CT butterfly, GS butterfly, and element-wise multiplier. For INTT, the scaling factor is incorporated into the butterfly unit instead of the end.	Optimized Barrett reduction [53] based on [84]
[85]	Kyber	Matrix-based NTT/INTT with 4×8 butterfly units utilizing RISC-V Instruction Set Architecture	FPGA: Xilinx Artix-7 AC701	Iterative CT and GS, performed with 4×8 butterfly units.	Not explained in the article.
[86]	Kyber	Optimizing the modular reduction part of NTT/INTT by implementing look-up tables.	FPGA: Zynq UltraScale+ ZCU1104	Iterative CT/GS with a singular butterfly unit.	Barrett reduction optimized with look-up tables.
[87]	Kyber	Matrix-based NTT/INTT utilizing Tensor Core, a feature to calculate matrix multiplication, part of NVIDIA AI Accelerator.	NVIDIA GeForce RTX 3080	Iterative CT and GS, parallelized as matrices to accommodate faster computation using Tensor Core.	Naive modular lazy reduction utilizing the fact that the results will not cause overflow in 32-bit integer representation.
[58]	Kyber	NTT/INTT coefficients are divided into two parts processed simultaneously.	FPGA: Xilinx Virtex-7	Parallelization called *Dual-path delay feedback (DDF)* to implement CT and GS butterfly.	Barrett and Montgomery reduction for different parts of the architecture.
[88]	Kyber	Low complexity butterfly unit and optimization of modular reduction based on K-RED [76]	FPGA: Intel Cyclone V 5CSXFC6D6F31C6	CT/GS butterflies with 2×2 configuration	Bitwise add-shift operations, based on K-RED [76]
[89]	Kyber	Optimized NTT/INTT operation specifically for the ARMv8 using NEON instruction set by vectorization.	Google Pixel 3 Android smartphone with 4 cores of ARM Cortex-A53 and 4 cores of ARM Cortex-A57	Naive CT/GS using matrix multiplication utilizing NEON instruction sets from the targeted device.	Montgomery and Barrett reduction for different parts of the design.
[90]	Kyber	Running Kyber on an IoT device using NEON-Optimized NTT proposed on [89].	Orange Pi with ARM Cortex-A53 Raspberry Pi 4 with ARM Cortex-A72	Naive CT/GS butterflies utilizing matrix operations provided by NEON instruction sets, based on [89].	Montgomery and Barrett reduction for different usage, both implemented using NEON Engine from ARMv8 architecture.
[91]	Kyber	Enabling lazy reduction (modular reduction in the end, instead of during calculation) by implementing Plantard arithmetic in the NTT/INTT calculation.	STM32F4 Discovery board with ARM Cortex-M4	Naive CT/GS butterflies using matrix operations provided by the target device instruction sets.	Optimization of Barrett and Montgomery reduction based on Plantard arithmetic [92].

TABLE 7. *(Continued.)* **Summary of Dilithium, Kyber, and Falcon's NTT hardware implementations.**

Ref.	Implemented PQC scheme	Novelty claim on NTT/INTT implementations	Target device	NTT/INTT algorithms	Modular reduction
[93]	Kyber	Designed a pipelined NTT/INTT with the flexibility of input parameters, such as using modulus q as input, not as a fixed parameter.	FPGA: Xilinx Virtex-7	Pipelined iterative CT/GS butterflies with flexible input parameters. For INTT, the scaling factors are incorporated into the twiddle factors.	Bitwise add-shift operations, named K-RED based on [76].
[94]	Kyber	Designed a 2×2 butterfly block called *Bi-core* that can act as CT or GS butterfly and element-wise multiplier.	FPGA: Xilinx Artix-7	Iterative CT/GS using 2×2 butterfly configuration.	Barrett reduction implemented with bitwise operations specifically for 24-bit input.
[95]	Kyber	Adopting R2MDC [71] and improved the design into R2^2MDC while also implementing pipelined architecture.	FPGA: Xilinx UltraScale+	Iterative CT/GS with radix-2, later optimized with radix-4.	Barrett reduction, modified with only add and shift operation.
[96]	Kyber	Configurable butterfly that supports both CT and GS.	65-nm TSMC cell library.	Iterative CT/GS with two butterfly cores employed in parallel.	Barrett reduction.
[97]	Kyber	Compact scheduling and sampling of NTT process and also unified butterfly for CT, GS, and point-wise multiplication	FPGA: Xilinx Artix-7	Iterative CT/GS with the scaling factor of INTT incorporated into twiddle factors.	Barrett reduction implemented with bitwise operations.
[98]	Kyber	Designed Instruction Set Extension for RISC-V specifically for NTT/INTT.	FPGA: Xilinx ZCU106 evaluation board.	Iterative CT/GS will be used in RISC-V instruction set extension.	Barrett and Montgomery reduction for different design parts.
[99]	Kyber	Reconfigurable datapath that can perform CT/GS butterflies and pointwise multiplication.	FPGA: Xilinx Artix-7	Iterative CT/GS with four butterfly units.	Bitwise operations specifically designed for $q = 3329$
[100]	Kyber	Designed the Kyber implementation efficiently for mobile devices using NEON instruction sets.	Raspberry Pi 3 with ARM Cortex-A53 processor.	Iterative CT/GS butterflies with the number of input $n = 16$ utilizing NEON instruction sets.	Barrett and Montgomery reduction optimized for the use of NEON instruction sets.
[101]	Kyber	Designed various polynomial multiplications for PQC schemes, including Kyber, utilizing NEON instruction sets from ARMv8 processor.	Raspberry Pi 4 with ARM Cortex-A72 processor	Iterative CT/GS with seven layers with detailed storage management for each layer.	Barrett reduction.
[102]	Kyber	Designed an 8-point butterfly unit that uses CT configuration in the first half while using GS configuration in the second half.	FPGA: Xilinx Artix-7	Iterative CT/GS manipulated so that CT is used in the first half, while GS is used in the second half to reduce hardware resources.	Bitwise operations based on [103].
[104]	Kyber	Reconfigurable butterfly unit that can act as CT or GS butterfly.	FPGA: Xilinx Artix-7	Iterative CT/GS with 2×2 butterfly unit.	Bitwise operations based on K-RED [76].
[105]	Kyber	Reconfigurable butterfly unit that can be used for CT butterfly, GS butterfly, and pointwise multiplication.	FPGA: Xilinx Artix-7	Iterative CT/GS butterflies while optimizing the storage management.	Barrett reduction implemented with bitwise operations.
[106]	Kyber	Parallelizing CT/GS butterflies using 8 butterfly units.	FPGA: details not referenced in the article	Iterative CT/GS with 8 butterfly units.	Barrett reduction implemented with bitwise operations.
[107]	Kyber	Pipelined GS-only butterflies structure for reducing the area consumption.	FPGA: Xilinx Kintex-7	Iterative GS butterflies for both NTT and INTT.	Barrett reduction based on [103]
[108]	Kyber	Pipelined butterfly arithmetic unit and optimized modular reduction	FPGA: Xilinx Artix-7 XC7A35TFTG256-1	Iterative CT/GS with optimized storage management (called *ping-pong*) in them.	Bitwise add-shift operations complemented with look-up tables.
[109]	Kyber	Efficient parallel and pipelined NTT/INTT module computation.	FPGA: Xilinx Artix-7 XC7A200T and Virtex-7 XC7VX485T	Iterative CT/GS with 4 coefficients processed simultaneously	Montgomery reduction.
[110]	Kyber	Implemented and optimized GS butterflies for both NTT and INTT with dual column sequential storage management	FPGA: Xilinx Artix-7 XC7A200T and Virtex-7 XC7VX485T	Iterative GS only butterflies for NTT/INTT with 4 coefficients processed at the same time.	Barrett reduction.

TABLE 7. *(Continued.)* **Summary of Dilithium, Kyber, and Falcon's NTT hardware implementations.**

Ref.	Implemented PQC scheme	Novelty claim on NTT/INTT implementations	Target device	NTT/INTT algorithms	Modular reduction
[111]	Kyber	Precomputing twiddle factors in Montgomery domain, the use of signed representation to prevent integer overflow.	STM32F4 Discovery board with ARM Cortex-M4	Iterative CT and GS using 2×2 configuration.	Montgomery reduction.
[112]	Falcon	Utilizing NEON engine in ARMv8 by parallelizing the overall process and optimizing memory access management.	Jetson Xavier board (ARMv8.2 8-cores CPU)	Iterative CT/GS butterflies utilizing vector operations by NEON engine.	Montgomery reduction.
[113]	Falcon	Parallelization of the operations in Falcon using GPU, including NTT/INTT parts.	Platform 1: Intel i9019899K CPU (3.70 GHz clock) with RTX 3080 GPU and 32 GB RAM; Platform 2: ARDC Nectar Research Cloud system [114] with A100, T4, and V1000 GPUs	Iterative CT/GS parallelized using 128 GPU threads.	Montgomery reduction.
[115]	Falcon and Dilithium	Hardware and Software co-design for both Dilithium and Falcon in one module with an NTT/INTT accelerator.	Pulpino Microcontroller [116]	Iterative CT/GS butterflies.	Not explained in the article.
[117]	Falcon	Designed a RISC-V architecture extension to support NTT/INTT operation.	FPGA: Xilinx Virtex-7 XC7VX690tffg1761-2	Iterative CT/GS butterflies using 2×2 configuration.	Montgomery reduction.

TABLE 8. **CPU, GPU, and FPGA comparison.**

Characteristic	CPU	GPU	FPGA
Definition	Central Processing Unit	Graphics Processing Unit	Field Programmable Gate Array
Cores	Several Cores	Huge Amount of Cores	Based on The Architecture
Clocks	Low Latency	High Throughput	High Throughput
Specialization	Serial Processing	Parallel Processing	Both
Power	High	Very High	Low
Programming	Low	Medium	Sophisticated

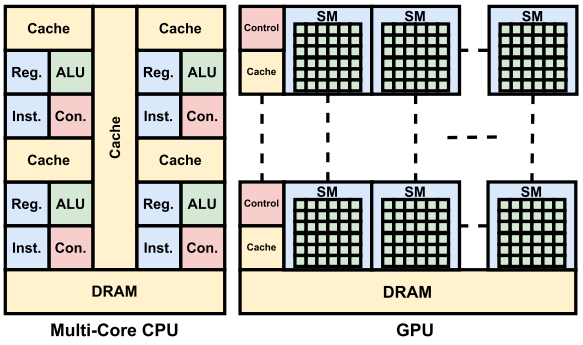


FIGURE 11. **CPU vs. GPU architecture.**

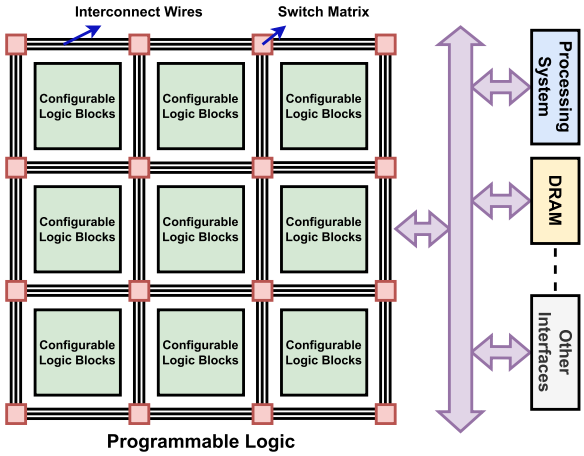


FIGURE 12. **FPGA architecture.**

The feature of the GPU is very suitable for processing the homomorphic encryption that requires polynomial operations with high degrees.

A field programmable gate array (FPGA) is a reconfigurable device that can perform specific functions or tasks. The biggest advantage of an FPGA is the ability to be reprogrammed with higher flexibility than CPUs and GPUs. As shown in Figure 12, the FPGA consists of configurable logic blocks and interconnects that can be changed by loading a bit stream onto the FPGA. An FPGA is suitable for applying high-performance cryptography, such as homomorphic encryption, because of its ability to perform computations in parallel and greater flexibility in embedding special functions on the hardware. Moreover, it also allows us to add some interfaces such as memory, Ethernet, video, and audio interfaces. Finally, the comparisons of CPU, GPU, and FPGA are shown in Table 8.

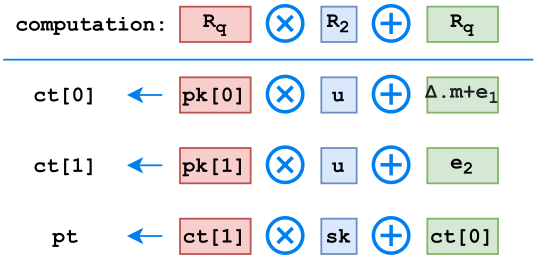


FIGURE 13. Computation pattern [46].

A. OVERVIEW OF HOMOMORPHIC ENCRYPTION

First, we explain the basic properties used in homomorphic encryption. $\mathbb{Z}[x]$ is a polynomial usually represented in integers. Thus, $\mathbb{Z}_q[x]$ is a set of integers $[0, q)$ where q is the coefficient modulus. Second, we have a ring of polynomial modulus is $\mathbb{R}_q = \mathbb{Z}_q[x]/\phi_m(x)$, where $\phi_m(x)$ is a cyclotomic polynomial in the form of $(x^n + 1)$ with n as the polynomial degree.

1) ENCRYPTION IN HE

Definition 6.1: The following represents the encryption of FHE [46]:

$$(c_0, c_1) = ([\Delta \cdot m + p_0 \cdot u + e_1]_q, [p_1 \cdot u + e_2]_q) \quad (30)$$

where c_0 and c_1 are the ciphertexts with errors $e_1, e_2 \leftarrow \chi$, p_0 and p_1 are the public keys, $m \in \mathbb{R}_t$ is the message, $u \leftarrow \mathbb{R}_2^n$ is the sparse random polynomial, and $\Delta = q/t$ is the ratio between the ciphertext coefficient modulus and the plaintext coefficient modulus.

2) DECRYPTION IN HE

Definition 6.2: The following represents the decryption of FHE [46]:

$$m = \left[\left\lfloor \frac{[c_0 + c_1 \cdot s]_q}{\Delta} \right\rceil \right]_t \quad (31)$$

where c_0 and c_1 are the ciphertexts, s is the secret key, q is the ciphertext coefficient modulus, t is the plaintext coefficient modulus, and the ratio between the ciphertext coefficient modulus and the plaintext coefficient modulus is $\Delta = q/t$.

3) COMPUTATIONAL PATTERN AND ARCHITECTURE OF HE

In Equations (30) and (31), polynomial multiplications are performed in the $[\cdot]$ operator while polynomial additions are performed in the $[+]$ operator. Figure 13 shows a computational pattern proposed in [46]. We can see that the polynomial multiplication operations are used for public keys $(pk[0], pk[1])$ in the encryption and ciphertext $(ct[1])$ in the decryption.

Figure 14 shows an architecture of BFV homomorphic encryption. Note that the polynomial multiplication is performed in the ring polymult core while the adder module performs the polynomial addition after the polymult core.

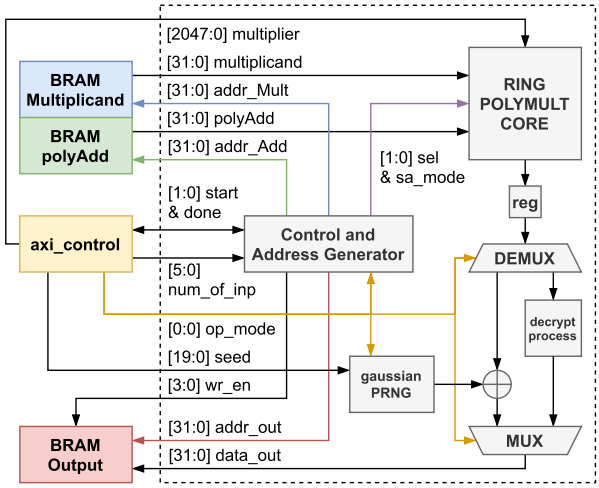


FIGURE 14. BFV accelerator architecture [46].

Moreover, a random number generator to generate sparse random polynomials and errors e_1 and e_2 is proposed as a Gaussian PRNG module.

Some polynomial operations are also used in homomorphic encryption, such as public key generation, relinearization, and bootstrapping. Some of them are computationally expensive, especially bootstrapping, which refreshes the noise in the ciphertext after homomorphic operations are performed. Bootstrapping includes evaluation and re-encryption of the ciphertext. They require polynomial multiplication and addition. Finally, the NTT module is implemented in the polynomial arithmetic cores to speed up the computation efficiently.

B. RLWE HOMOMORPHIC ENCRYPTION SCHEMES

In this subsection, we review some ring learning with errors (RLWE) based homomorphic encryption schemes *i.e.,* FHEW [119], CKKS [39], BFV [36], BGV [37], [38], and TFHE [120]. One of the pioneers of HE is the Fully Homomorphic Encryption over the Weil Descent (FHEW), RLWE-based homomorphic encryption founded by Gentry and Halevi, which combines RLWE and Weil descent techniques to achieve homomorphic operations on ciphertexts. FHEW is designed to be efficient in computation and memory usage, making it suitable for resource-constrained environments.

The CKKS (Cheon-Kim-Kim-Song) scheme [39] is a Leveled Homomorphic Encryption scheme designed to support real-valued data in complex numbers. It is well-suited for data with a large dynamic range and is sensitive to numerical errors, such as the data from machine learning models. CKKS uses a polynomial ring to represent encrypted data and employs the relinearization technique to reduce computational complexity. It is known for its fast performance on real-world applications, such as machine learning inference.

The BFV (Brakerski-Gentry-Vaikuntanathan) scheme [36] is a type of FHE with integer representations. Next, the BGV (Brakerski-Gentry-Vaikuntanathan) scheme [37], [38] is a type of FHE designed to work with integers and binary data. It is also well-suited for modular arithmetic and logical operations applications, such as privacy-preserving data analysis. BGV uses a polynomial ring to represent encrypted data and employs bootstrapping to reduce computational complexity. Bootstrapping enables the FHE scheme to evaluate complex functions on encrypted data by recursively re-encrypting ciphertexts. BGV is known for its ability to handle more complex computations than BFV.

The TFHE (Fully Homomorphic Encryption from Torus) scheme [120] is a somewhat homomorphic encryption (SHE) scheme that is based on the GSW scheme [121] and the ring-learning with errors (RLWE) problem. It is designed to be efficient in computation and memory usage. TFHE is a single-key scheme, meaning the same key is used for encryption, decryption, and homomorphic operations. It is also designed to be fully compatible with standard Boolean circuits, making it suitable for various applications.

C. HOMOMORPHIC ENCRYPTION IMPLEMENTATIONS IN VARIOUS PLATFORMS

1) HE IMPLEMENTATIONS IN CLASSICAL CPU

Some software for homomorphic encryption is HELib [122], SEAL [41], OpenFHE [123], and LATTIGO [124]. First, HELib [122] implements a lattice polynomial ring using a C++ library that consists of BGV and CKKS schemes. It supports high-performance computing environments with parallelization, optimized arithmetic operations, and memory management. Second, Microsoft SEAL [41] implements BFV, BGV, and CKKS. It is a well-known software that efficiently improved by implementing the residue number system (RNS) FV. SEAL also supports various parameters. Next, PALISADE (Privacy-Enhancing Technologies: Algorithms, Implementations, and Development Environments), now called OpenFHE [123], is a modular C++ library for FHE that supports multiple schemes, including CKKS, BGV, and the Fan-Vercauteren (FV) scheme. It is designed to be extensible and includes various features like multithreading, GPU acceleration, and network communication. Finally, LATTIGO [124] supports CKKS and BGV implemented in the Golang library. It is fast and designed for ease of use. It includes features like automatic parameter selection, key management, and ciphertext packing and provides an interface for easy integration with machine learning frameworks like TensorFlow.

Next, we explore various NTT implementations in GPUs and FPGAs as hardware accelerators. As explained before, the advantages of GPU and FPGA over CPU are in the flexibility, configurability, and performance due to massive cores and specific functions. We will see how parallel architecture is applied in fast large integer arithmetics by the residual number system and the transformer, in this case, butterfly operations in NTT.

2) HE IMPLEMENTATIONS IN GPUs

Some works in GPUs for homomorphic encryption are presented in [125], [126], [127], [128], [129], and [130]. The optimizations implemented in GPUs can be classified into 3 major optimizations. The first is the parallel implementation of butterfly modules in the NTT since GPU is capable of massively parallel computing [125], [126], [130], [131]. The second is optimizing the kernel and memory management [125], [126], [130], [131]. And the last is the optimization of modular arithmetic operations, such as the Barret multiplication, the Montgomery multiplication, and the Residual Number System (RNS) based on the Chinese Remainder Theorem (CRT) [125], [126], [128], [129], [130], [131].

Figure 15 depicts the implementation of HE-Booster. It consists of 5 phases to do the homomorphic encryption operations [125]. The first phase is CRT which decomposes ciphertext into multiple independent sub-space. The coefficient reduction operation provides a higher utilization of GPU parallel architecture. Second, NTT is performed by using the CT butterfly. It uses inter-thread local synchronization for optimization. Next, element-wise modular operations are performed in the third phase, dyadic computation. Fourth, using the GS butterfly, the INTT phase is the same mechanism as the second NTT phase. Finally, the inverse CRT phase reconstructs multiple residue polynomials in an independent sub-space into a single polynomial in ciphertext space.

Figure 16 shows the GPU thread strategy for executing the butterflies in the NTT [131]. In this case, the NTT operation uses 4 iterations with 16 inputs. In each iteration, there are 8 butterfly operations. Thus, we have the following:

Lemma 6.1: Suppose that we have n-input NTT/INTT. The serial implementation of NTT/INTT complexity is

$$O(\frac{n}{2}\log n).$$

Proof: It is clear that to achieve the last butterfly operation between input with index i and $i+1$, it requires $\log n$ iterations where there are $n/2$ butterfly operations in each iteration. □

Moreover, if we analyze the complexity of polynomial multiplication using NTT/INTT, we have the following:

Lemma 6.2: A polynomial multiplication of A and B using NTT/INTT that has n coefficients or inputs using serial computation takes at least:

$$O(n + 2n\log n).$$

Proof: A polynomial multiplication requires an NTT, a component-wise product, and an INTT for each A and B polynomial. An NTT/INTT takes $O(\frac{n}{2}\log n)$ while an element-wise operation takes $O(n)$. Thus, we have the lemma. □

Now, we define a quota of parallelization in the GPU as:

Definition 6.3: Suppose that C is the number of GPU cores, and h is the number of parallel polynomial operations executed in the GPU. The quota in executing a polynomial operation in a GPU is C/h cores.

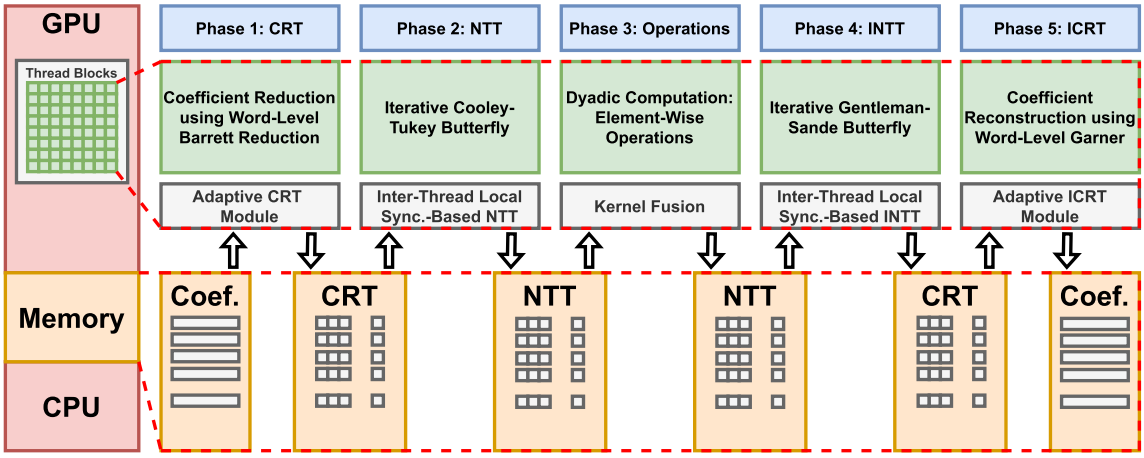


FIGURE 15. HE-Booster GPU Implementation [125].

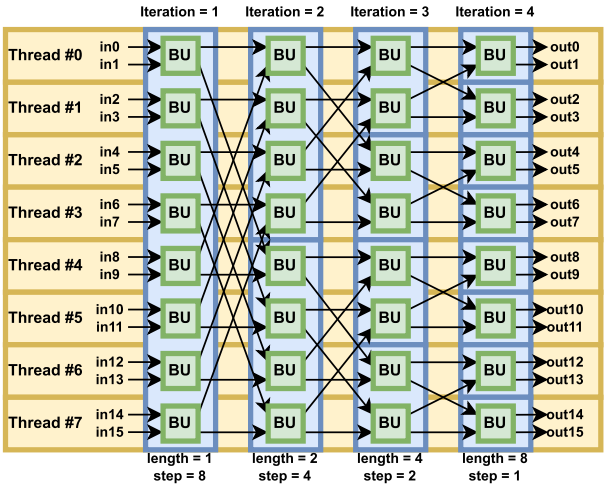


FIGURE 16. GPU thread assignment for $n = 16$ [131].

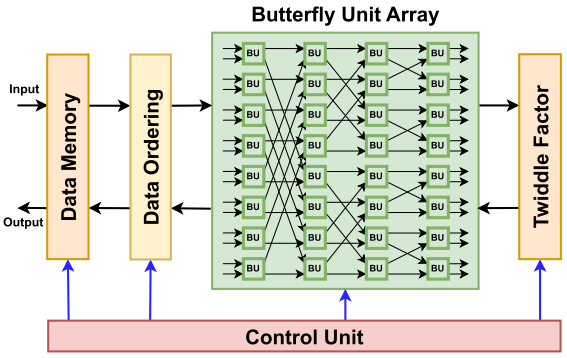


FIGURE 17. Parallel NTT-INTT architecture implementation [135].

Theorem 6.3: For encryption and decryption, the complexities for processing h plaintexts in parallel with C cores GPU are:

$$O(\frac{6nh}{C}(1+\log n)),$$

and

$$O(\frac{2nh}{C}(1+\log n)),$$

respectively, where n is the number of inputs or coefficients in NTT/INTT.

Proof: As Lemmas 6.1 and 6.2, encryption requires at most 3 polynomial multiplications and 3 polynomial additions, while decryption requires a polynomial multiplication and a polynomial addition. With C-core GPU and h parallel processes, we have the theorem. □

By targeting high utilization of a GPU, if $6nh$ is less than C, we will have a very fast $O(\log n)$ execution time. Thus, n, h, and C determine the system's scalability. The calculation is also applied for FPGA implementations with parallel processing elements.

3) HE IMPLEMENTATIONS IN FPGAs

On the other hand, FPGAs also offer efficient area and speed implementation for homomorphic encryption (HE) presented in [117], [127], [132], [133], [134], [135], [136], [137], [138], and [139]. The implementation of HE in FPGA also consists of 3 approaches. First, pre/post-processing is usually required in the NTT/INTT by using the RNS-CRT method and modular multipliers such as Barrett, Montgomery, or LUT-based reduction [127], [132], [133], [134], [137], [139]. The second is the parallel implementation of processing elements (PEs) or butterfly units (BUs), in serial or pipeline parallelization [127], [132], [133], [134], [135], [138], [139]. And the last, the optimization is done by reconfigurable designs by implementing custom PEs or instructions using RISC-V [117], [132], [138], [139].

Figures 17 and 18 show the NTT/INTT architecture implemented in FPGAs. In [135], the butterfly unit array is constructed with 8×4 arrays as shown in Figure 17. The architecture is capable of transforming 16 coefficients

TABLE 9. Each modulus calculation of NWC via NTT/INTT.

q	ψ	A	B	$NTT^{\psi}(A)$	$NTT^{\psi}(B)$	$\hat{C} = NTT^{\psi}(A) \circ NTT^{\psi}(B)$	$INTT^{\psi^{-1}}(\hat{C})$
6841	3095	[318,3338,5017,1115]	[4054,152,6672,2770]	[389, 5541, 3394, 5630]	[2098, 3348, 4040, 6730]	[2043, 5317, 2396, 4442]	[129, 1912, 6335, 3594]
7681	1925	[560,2624,6099,5678]	[5257,4836,1995,1574]	[1856, 3599, 2766, 1700]	[5978, 5942, 7189, 1919]	[3804, 1354, 6346, 5556]	[4265, 7029, 1887, 2848]
8681	4219	[1922,8663,8461,4040]	[8300,3879,6719,2298]	[712, 771, 1130, 5075]	[6470, 7581, 2693, 7775]	[5710, 2638, 4740, 2980]	[4017, 8106, 6657, 2055]

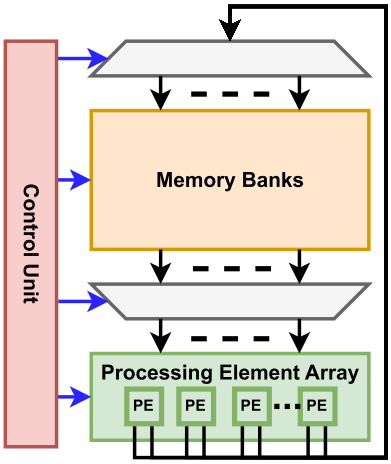


FIGURE 18. NTT-INTT architecture implementation with a Custom RISC-V Instructions [138].

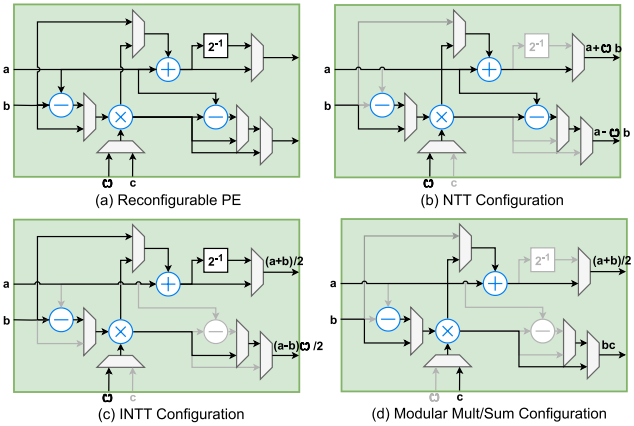


FIGURE 19. ReMCA architecture implementation for NTT, INTT and modular operations [139].

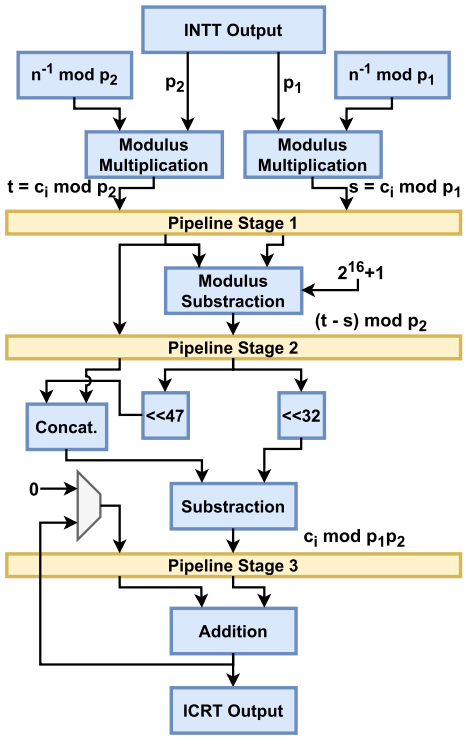


FIGURE 20. ICRT architecture implementation [136].

in the pipeline. The work is targeted for the RNS-CKKS scheme.

In [138], NTT/INTT architecture is proposed with custom instructions of Linux-ready RISC-V core. The number of processing elements (PE) is varied. The BRAMs have three parameters *i.e.,* depth, width, and number of PEs. Increasing the number of PEs reduces the depth of the memories. However, more memories require to be initiated. The architecture in Figure 18 uses one-dimensional PE, while the architecture in Figure 17 uses two-dimensional PEs in the form of butterfly units (BUs). Regarding the speed of execution, the architecture in [135] is better, while in terms of programmability and area, the architecture in [138] is better. Another work [117] also proposes a RISC-V architecture extension for NTT without additional hardware modification.

Figure 19 shows a unified dynamic reconfigurable that can be flexible into several modes *i.e.,* a butterfly unit for NTT (CT-form), a butterfly unit for INTT (GS-form), and a modular multiplication. The work implements a BFV algorithm with NTT/INTT pre/post-processing with RNS.

From Subsection III-E we know that we can choose a composite modulus instead of a prime. A large composite modulus can be factored into a product of small primes q_i such that $\prod_{i=0}^{l-1} q_i$, where l is the number of small primes. An example implementation of RNS based on CRT is proposed in [136] as depicted in Figure 20. It reconstructs modulo p_1 and p_2 into a larger modulo p_1p_2. Note that for reconstructing a modulo from two smaller modulos, it requires 3 pipeline stages.

The biggest advantage of breaking down into RNS representations is each NTT/INTT modulus can be calculated independently without any dependency on one another. This advantage makes it easy to parallelize NTT/INTT with a composite modulus. We provide a toy example of using NTT, RNS, and CRT combinations in Example 9.

TABLE 10. Summary of NTT hardware implementations for homomorphic encryption.

Ref.	Implemented HE scheme	Novelty claim on NTT/INTT implementations	Target device	NTT/INTT algorithms	Modular reduction
[126]	Only focused on the polynomial multiplication part.	Breaking down NTT/INTT using RNS-CRT, then parallelizing them using several GPU kernels.	NVIDIA GPU with CUDA programming model.	Iterative CT/GS butterflies with 4096-point along with parallelization in GPU.	Montgomery reduction.
[127]	NTRU-based FHE schemes.	Breaking down NTT/INTT using RNS-CRT and optimizing the modular reduction into hardware-friendly operations.	FPGA: Virtex-7 XC7VX690T	Iterative CT/GS butterflies with 4×4 butterfly units.	Barrett reduction using hardware-friendly operations.
[131]	BFV Homomorphic Encryption	Modifying NTT/INTT and RNS-CRT algorithm structure to make it parallel-friendly to be implemented in GPU.	NVIDIA Tesla V100 GPU	Iterative CT/GS modified to consist of three *for* loops without dependencies between them.	Barrett reduction.
[129]	NTRU-based Homomorphic Encryption	Building a GPU library to facilitate FHE. On the NTT/INTT part: breaking it down using RNS-CRT.	NVIDIA GeForce GTX 690 GPU	Iterative CT/GS butterflies with four steps.	Not explained in the article.
[125]	Five phases in typical FHE schemes.	Breaking down NTT/INTT using RNS-CRT and parallelizing them using single and multiple GPUs.	NVIDIA GeForce RTX3070 GPU	Iterative CT/GS butterflies parallelized in GPU.	Barrett reduction.
[130]	BFV Scheme	Breaking down NTT/INTT using RNS-CRT and parallelization using GPU.	NVIDIA GPU with CUDA programming model	Iterative CT/GS butterflies parallelized in GPU with memory management.	Barrett reduction.
[140]	BFV Scheme	Breaking down NTT/INTT using RNS-CRT, implemented using multithreading by OpenMP in CPU and parallelization in GPU.	NVIDIA Tesla K80 and V100	Iterative CT/GS butterflies with loop parallelization.	Barrett reduction.
[141]	CNT FHE scheme	Low Hamming weight and parallelized NTT/INTT architecture with concerns to minimize latency and hardware resource usage.	FPGA: Xilinx Virtex-7	Iterative CT/GS butterflies with low hamming weight parameters.	Bitwise operations utilizing the chosen modulus property.
[132]	Only focused on the polynomial multiplication part.	Multi-core NTT/INTT architecture with reconfigurable processing elements and memory access management.	FPGA: Xilinx Virtex-7	Iterative CT/GS butterflies with the scaling factor distributed evenly to each layer of twiddle factors for INTT.	Barrett reduction.
[133]	Gentry-Halevi (GH) FHE scheme	Implementing all GH FHE primitives, including NTT and INTT.	ASIC: No further details	Iterative CT/GS with 12×12 butterfly units.	Barrett reduction.
[136]	Focused on large integer multiplication.	Factoring their composite modulus q into two other moduli and merging the results using CRT.	FPGA: Altera Stratix-V 5SGXEA4H1F35C2	Iterative GS for both NTT and INTT with radix-16 butterfly.	Bitwise add-shift operations based on their chosen moduli.
[142]	FV Homomorphic Encryption	Breaking down NTT/INTT using RNS-CRT and process them in parallel with a designed unit called *Residue Parallel Arithmetic Unit (RPAU)*	FPGA: Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit	Iterative CT/GS parallelized into two cores with memory access management during parallelization.	Sliding window and lookup tables.
[137]	Focused on large integer multiplication.	Comparing NTT/INTT multiplication with other schemes, such as Karatsuba, Comba, and their combinations.	FPGA: Xilinx Virtex-7	Iterative CT/GS butterflies.	Bitwise operations based on the chosen modulus.
[139]	BFV Homomorphic Encryption.	Breaking down NTT/INTT using RNS-CRT; designed reconfigurable butterfly unit that can be used as CT/GS butterfly.	FPGA: Xilinx Virtex-7	Iterative CT/GS butterflies with parallelization and memory access management.	Barrett reduction.
[134]	Focused on the implementations of NTT/INTT only.	Comparing various hardware design methods: parametric hardware design, high-level synthesis, and hardware-software co-design for NTT/INTT.	FPGA: Xilinx Virtex-7	Iterative CT/GS with memory access management.	Montgomery reduction.
[138]	Focused on the NTT/INTT architecture.	Extending RISC-V Instruction Set Architecture with modular arithmetics to support NTT/INTT implementation.	FPGA: Xilinx Ultrascale+	Iterative CT/GS butterflies breaking down using RNS-CRT.	Montgomery reduction.
[135]	CKKS HE scheme.	Area-efficient NTT/INTT architecture with twiddle factors generated on the fly.	FPGA: Xilinx Zynq UltraScale+ ZCU102	Iterative CT/GS butterfly with 8×4 butterfly units.	Barrett reduction.

Example 6.1: Let $\boldsymbol{A} = [123456, 7891011, 121314, 151617]$ *and* $\boldsymbol{B} = [181920, 212223, 232425, 262728]$ *in the ring* $\mathbb{Z}_Q$ *with* $Q = 456149404001$. *We need to find the negacyclic convolution of* $\boldsymbol{A}$ *and* $\boldsymbol{B}$.

Calculating the results using previously explained methods in normal order: *Using* $\psi = 12967992388$, *we can get*:

$$NTT^{\psi}(\boldsymbol{A}) = [164909637252, 371837718802, 52022178059, 323529767713]$$
$$NTT^{\psi}(\boldsymbol{B}) = [94256621661, 54777633553, 418495999451, 344769281017]$$

Element-wise multiplication between the two yields:

$$NTT^{\psi}(\boldsymbol{A}) \circ NTT^{\psi}(\boldsymbol{B}) = [164909637252, 371837718802, 52022178059, 323529767713]$$

Taking INTT from the results yields in the negacyclic convolution between $\boldsymbol{A}$ *and* $\boldsymbol{B}$

$$INTT^{\psi^{-1}}(NTT^{\psi}(\boldsymbol{A}) \circ NTT^{\psi}(\boldsymbol{B})) = [169643576476, 26172545988, 317135487954, 95233749301]$$

Therefore the negacyclic convolution between them is [169643576476, 26172545988, 317135487954, 95233749301].

Calculating the results using RNS and CRT: *Notice that the modulus* Q *is a composite number that can be factored into three primes* $Q = q_1 \times q_2 \times q_3$, *where* $q_1 = 6841$, $q_2 = 7681$, *and* $q_3 = 8681$. *We can use the modulus factors as moduli set for a residue number system:* $\{6841, 7681, 8681\}$. *We can represent each element in* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *in the RNS representation. Take the element 123456 as an example*:

$$123456 \equiv 318 \mod 6841,$$
$$123456 \equiv 560 \mod 7681,$$
$$123456 \equiv 1922 \mod 8681$$

Therefore 123456 *can be represented as* (318, 560, 1922) *in the RNS representation based on our chosen moduli. Hence, transforming* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *in the RNS representation in our chosen moduli*:

$$\boldsymbol{A} = [(318, 560, 1922), (3338, 2624, 8663), (5017, 6099, 8461), (1115, 5678, 4040)]$$
$$\boldsymbol{B} = [(4054, 5257, 8300), (152, 4836, 3879), (6672, 1995, 6719), (2770, 1574, 2289)]$$

NTT-INTT can calculate the negacyclic convolution for each modulus in the set. Table 9 shows the calculation details.

From the last column, we got the RNS representation of the negacyclic convolution between $\boldsymbol{A}$ *and* $\boldsymbol{B}$:

$$\boldsymbol{C} = [(129, 4265, 4017), (1912, 7029, 8106), (6335, 1887, 6657), (3594, 2848, 2055)]$$

Converting each element back to normal representation can be done using the Chinese Reminder Theorem. Take the element (129, 4265, 4017) *as an example, we need to find x from the following system of equations:*

$$x \equiv 129 \mod 6841,$$
$$x \equiv 4265 \mod 7681,$$
$$x \equiv 4017 \mod 8681$$

This is a classical textbook problem for CRT. Solving it, we can obtain $x = 169643576476$. *Transforming back all* $\boldsymbol{C}$ *to normal representation yields*:

$$\boldsymbol{C} = [169643576476, 26172545988, 317135487954, 95233749301]$$

Which is the same result as the negacyclic convolution when calculated directly.

D. IMPLEMENTATION COMPARISON

Finally, to close this section, Table 10 summarizes the implementation comparisons between various researchers in the HE schemes based on their novelty claim on NTT/INTT implementations algorithm, target device or hardware, their presented NTT/INTT implementations, and how they implement the modular reduction.

VII. CONCLUSION AND FUTURE WORKS

A. CONCLUSION

We reviewed the concepts of Number Theoretic Transform (NTT) and its inverse (INTT). We also provided a comprehensive survey about their implementation in the standardized Post-Quantum Cryptographic (PQC) scheme by the NIST and in Homomorphic Encryption (HE). In summary, we conclude that:

1) We comprehensively introduced the concepts of NTT/INTT and the other concepts surrounding it. Many other pieces of literature briefly introduce the concepts, but they are scattered everywhere, requiring significant effort to learn. Our report should be helpful, especially for those who begin researching the area and come from the engineering or implementation side.
2) We provided consistent, small, and understandable toy examples through different concepts and algorithms to further enhance the conceptual understanding of the NTT/INTT, which hopefully helps in understanding the concepts.
3) We summarized and provided a comprehensive review of the recent research on the NTT/INTT implementations for Post Quantum Cryptography (PQC) schemes in various platforms such as FPGAs, GPUs, and various embedded systems.
4) Similarly, we also summarize and provide a comprehensive review of another use case of NTT/INTT implementations for Homomorphic Encryption (HE) schemes, including its optimizations, such as the combinations of NTT-RNS-CRT for its parallelizations.

B. FUTURE WORKS

While NTT/INTT is useful for many applications in Post-Quantum Cryptography and Homomorphic Encryption, most researchers currently do not consider the secure implementation of NTT/INTT. Side-channel attacks have increasingly become a concern because they can use leaked information to recover some secrets of cryptographic schemes.

One research suggests that all operations should be under the strategy of constant implementation to avoid timing attacks [57]. There are also known types of attacks in NTT/INTT implementation, including single trace attacks, simple power analysis, and fault attacks [4], [143], [144].

Therefore, we suggest in the future, researchers also need to consider the secure implementation of NTT/INTT.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, Nov. 1994, pp. 124–134.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41, no. 2, pp. 303–332, Jan. 1999.

[3] J. A. Buchmann, D. Butin, F. Göpfert, and A. Petzoldt, "Post-quantum cryptography: State of the art," in *The New Codebreakers: Essays Dedicated to David Kahn Occasion His 85th Birthday*. Berlin, Germany: Springer, 2016, pp. 88–108.

[4] H. Nejatollahi, N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota, "Post-quantum lattice-based cryptography implementations: A survey," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–41, Nov. 2019.

[5] L. Chen, D. Moody, and Y. Liu, "NIST post-quantum cryptography standardization," *Transition*, vol. 800, no. 131A, p. 164, 2017.

[6] G. Alagic, "Status report on the first round of the NIST post-quantum cryptography standardization process," 2019.

[7] G. Alagic, *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*, vol. 2. Gaithersburg, MD, USA: U.S. Department of Commerce, NIST, 2020.

[8] G. Alagic, *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. Gaithersburg, MD, USA: U.S. Department of Commerce, NIST, 2022.

[9] NIST. *Selected Algorithms 2022*. Accessed: Mar. 8, 2023. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

[10] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium: A lattice-based digital signature scheme," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, pp. 238–268, Feb. 2018.

[11] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai, "CRYSTALS-Dilithium," in *Algorithm Specifications Supporting Documentation*, 2020.

[12] P.-A. Fouque, "FALCON: Fast-Fourier lattice-based compact signatures over NTRU," in *Submission to NIST's Post-Quantum Cryptography Standardization Process*, vol. 36, no. 5, 2018.

[13] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS–Kyber algorithm specifications and supporting documentation (version 1.0)," *NIST Post-Quantum Cryptogr. Standardization Process*, vol. 2, no. 4, pp. 1–43, 2017.

[14] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS—Kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 353–367.

[15] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS–Kyber algorithm specifications and supporting documentation (version 2.0)," *NIST Post-Quantum Cryptogr. Standardization Process*, to be published.

[16] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS–Kyber algorithm specifications and supporting documentation (version 3.0)," *NIST Post-Quantum Cryptogr. Standardization Process*, to be published.

[17] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, and G. Gogniat, "Exploration of polynomial multiplication algorithms for homomorphic encryption schemes," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2015, pp. 1–6.

[18] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers," *Doklady Akademii Nauk SSSR*, vol. 145, no. 2, pp. 293–294, 1962.

[19] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba algorithm for efficient implementations," *Cryptol. ePrint Arch.*, to be published.

[20] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," *Soviet Math. Doklady*, vol. 3, no. 4, pp. 714–716, 1963.

[21] S. A. Cook and S. O. Aanderaa, "On the minimum computation time of functions," *Trans. Amer. Math. Soc.*, vol. 142, pp. 291–314, Aug. 1969.

[22] E. Chu and A. George, *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*. Boca Raton, FL, USA: CRC Press, 1999.

[23] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, 1965.

[24] W. M. Gentleman and G. Sande, "Fast Fourier transforms: For fun and profit," in *Proc. November 7–10, Fall Joint Comput. Conf. XX AFIPS (Fall)*, 1966, pp. 563–578.

[25] T. Jebelean, "Practical integer division with Karatsuba complexity," in *Proc. Int. Symp. Symbolic Algebr. Comput. (ISSAC)*, 1997, pp. 339–341.

[26] M. Bodrato, "Towards optimal Toom–Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0," in *Proc. Int. Workshop Arithmetic Finite Fields, (WAIFI)*, Madrid, Spain: Springer, Jun. 2007, pp. 116–133.

[27] R. C. Agarwal and C. S. Burrus, "Number theoretic transforms to implement fast digital convolution," *Proc. IEEE*, vol. 63, no. 4, pp. 550–560, Apr. 1975.

[28] H. Nussbaumer, "Fast polynomial transform algorithms for digital convolution," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-28, no. 2, pp. 205–215, Apr. 1980.

[29] J. M. Pollard, "The fast Fourier transform in a finite field," *Math. Comput.*, vol. 25, no. 114, pp. 365–374, 1971.

[30] C. Gauss, *Theoria Interpolationis Methodo Nova Tractata Werke Band 3, 265–327*. Göttingen, Germany: Kónigliche Gesellschaft der Wissenschaften, 1886.

[31] D. Harvey and J. van der Hoeven, "Integer multiplication in time $O(n \log n)$," *Ann. Math.*, vol. 193, no. 2, pp. 563–617, Mar. 2021.

[32] E. W. Weisstein. (2015). *Fast Fourier Transform*. [Online]. Available: https://mathworld.wolfram.com/

[33] Z. Liang and Y. Zhao, "Number theoretic transform and its applications in lattice-based cryptosystems: A survey," 2022, *arXiv:2211.13546*.

[34] P. B. Modak, "Implementation of an RNS based sequential NTT convolver," Tech. Rep., 1982.

[35] H. Krishna, K.-Y. Lin, and B. Krishna, "Rings, fields, the Chinese remainder theorem and an extension—Part II: Applications to digital signal processing," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 41, no. 10, pp. 656–668, 1994.

[36] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptol. ePrint Arch.*, to be published.

[37] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-LWE and security for key dependent messages," in *Proc. Annual Cryptol. Conf.*, Santa Barbara, CA, USA: Springer, Aug. 2011, pp. 505–524.

[38] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, Jul. 2014.

[39] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Hong Kong: Springer, Dec. 2017, pp. 409–437.

[40] K. Laine and R. Player, "Simple encrypted arithmetic library-seal (V2. 0)," Tech. Rep., 2016.

[41] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-SEAL v2. 1," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Sliema, Malta: Springer, Apr. 2017, pp. 3–18.
[42] H. J. Nussbaumer, "Elements of number theory and polynomial algebra," in *Fast Fourier Transform and Convolution Algorithms*, 1982, pp. 4–31.
[43] *Convolution and Polynomial Multiplication in MATLAB*. Accessed: May 2, 2023. [Online]. Available: https://www.mathworks.com/help/MATLAB/ref/conv.html
[44] *Numpy Convolution*. Accessed: May 2, 2023. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.convolve.html
[45] *Modulo-n Circular Convolution in MATLAB*. Accessed: May 2, 2023. [Online]. Available: https://www.mathworks.com/help/signal/ref/cconv.html
[46] I. Syafalni, G. Jonatan, N. Sutisna, R. Mulyawan, and T. Adiono, "Efficient homomorphic encryption accelerator with integrated PRNG using low-cost FPGA," *IEEE Access*, vol. 10, pp. 7753–7771, 2022.
[47] *Sympy 1.11 Documentation*. Accessed: May 2, 2023. [Online]. Available: https://docs.sympy.org/latest/modules/ntheory.html#sympy.ntheory.residue_ntheory.nthroot_mod
[48] A. Schönhage and V. Strassen, "Schnelle multiplikation grosser Zahlen," *Computing*, vol. 7, nos. 3–4, pp. 281–292, 1971.
[49] V. S. Dimitrov, T. V. Cooklev, and B. D. Donevsky, "Generalized Fermat–Mersenne number theoretic transform," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 41, no. 2, pp. 133–139, Feb. 1994.
[50] C. M. Rader, "Discrete convolutions via Mersenne transrorms," *IEEE Trans. Comput.*, vol. C-100, no. 12, pp. 1269–1273, Dec. 1972.
[51] P. Heckbert, "Fourier transforms and the fast Fourier transform (FFT) algorithm," *Comput. Graph.*, vol. 2, p. 463, Feb. 1995.
[52] A. Saidi, "Decimation-in-time-frequency FFT algorithm," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Apr. 1994, p. 453.
[53] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Proc. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2000, pp. 311–323.
[54] T. Wu, S. Li, and L. Liu, "Modular multiplier by folding Barrett modular reduction," in *Proc. IEEE 11th Int. Conf. Solid-State Integr. Circuit Technol.*, Oct. 2012, pp. 1–3.
[55] L. Hars, "Long modular multiplication for cryptographic applications," in *Proc. CHES*. Cham, Switzerland: Springer, 2004, pp. 45–61.
[56] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, 1985.
[57] C. K. Koc, T. Acar, and B. S. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, Jun. 1996.
[58] T. T. Nguyen, S. Kim, Y. Eom, and H. Lee, "Area-time efficient hardware architecture for CRYSTALS–Kyber," *Appl. Sci.*, vol. 12, no. 11, p. 5305, May 2022.
[59] A. Abdulrahman, V. Hwang, M. J. Kannwischer, and A. Sprenkels, "Faster Kyber and Dilithium on the Cortex-M4," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Rome, Italy: Springer, Jun. 2022, pp. 853–871.
[60] F. Yaman, A. C. Mert, E. Öztürk, and E. Savas, "A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1020–1025.
[61] A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "KaLi: A crystal for post-quantum security using Kyber and Dilithium," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 2, pp. 747–758, Feb. 2023.
[62] Y. Kim, J. Song, T.-Y. Youn, and S. C. Seo, "CRYSTALS–Dilithium on ARMv8," *Secur. Commun. Netw.*, vol. 2022, pp. 1–12, Feb. 2022.
[63] X. Chen, B. Yang, Y. Lu, S. Yin, S. Wei, and L. Liu, "Efficient access scheme for multi-bank based NTT architecture through conflict graph," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, Jul. 2022, pp. 91–96.
[64] T. Wang, C. Zhang, P. Cao, and D. Gu, "Efficient implementation of Dilithium signature scheme on FPGA SoC platform," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 9, pp. 1158–1171, Sep. 2022.
[65] X. Chen, B. Yang, S. Yin, S. Wei, and L. Liu, "CFNTT: Scalable radix-2/4 NTT multiplication architecture with an efficient conflict-free memory mapping scheme," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2022, pp. 94–126, Nov. 2021.
[66] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, pp. 49–72, Mar. 2020.
[67] G. Mao, D. Chen, G. Li, W. Dai, A. I. Sanka, Ç. K. Koç, and R. C. C. Cheung, "High-performance and configurable SW/HW Co-design of post-quantum signature CRYSTALS-Dilithium," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 3, pp. 1–28, Sep. 2023.
[68] N. Gupta, A. Jati, A. Chattopadhyay, and G. Jha, "Lightweight hardware accelerator for post-quantum digital signature CRYSTALS-Dilithium," *IACR Cryptol. ePrint Arch.*, vol. 2022, p. 496, Jan. 2022.
[69] J. Zheng, F. He, S. Shen, C. Xue, and Y. Zhao, "Parallel small polynomial multiplication for Dilithium: A faster design and implementation," in *Proc. 38th Annu. Comput. Secur. Appl. Conf.*, Dec. 2022, pp. 304–317.
[70] C. Zhao, N. Zhang, H. Wang, B. Yang, W. Zhu, Z. Li, M. Zhu, S. Yin, S. Wei, and L. Liu, "A compact and high-performance hardware architecture for CRYSTALS-Dilithium," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2022, pp. 270–295, Nov. 2021.
[71] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proc. Int. Conf. Parallel Process.*, Apr. 1996, pp. 766–770.
[72] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 43, no. 10, pp. 677–688, Oct. 1996.
[73] G. Land, P. Sasdrich, and T. Güneysu, "A hard CRYSTAL—Implementing Dilithium on reconfigurable hardware," in *Proc. Int. Conf. Smart Card Res. Adv. Appl.* Lübeck, Germany: Springer, Nov. 2022, pp. 210–230.
[74] Z. Zhou, D. He, Z. Liu, M. Luo, and K.-K.-R. Choo, "A software/hardware co-design of CRYSTALS-Dilithium signature scheme," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 14, no. 2, pp. 1–21, Jun. 2021.
[75] D. T. Nguyen, V. B. Dang, and K. Gaj, "A high-level synthesis approach to the software/hardware codesign of NTT-based post-quantum cryptography algorithms," in *Proc. Int. Conf. Field-Program. Technol. (ICFPT)*, Dec. 2019, pp. 371–374.
[76] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Proc. Int. Conf. Cryptol. Netw. Secur.* Milan, Italy: Springer, Nov. 2016, pp. 124–139.
[77] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of CRYSTALS-Dilithium," in *Proc. Int. Conf. Field-Program. Technol. (ICFPT)*, Dec. 2021, pp. 1–10.
[78] K. D. Ortega L. and L. J. Dominguez Perez, "Implementing CRYSTAL-Dilithium on FRDM-K64," in *Proc. IEEE 12th Annu. Ubiquitous Comput., Electron. Mobile Commun. Conf. (UEMCON)*, Dec. 2021, pp. 178–183.
[79] S. Ricci, L. Malina, P. Jedlicka, D. Smékal, J. Hajny, P. Cibik, P. Dzurenda, and P. Dobias, "Implementing CRYSTALS-Dilithium signature scheme on FPGAs," in *Proc. 16th Int. Conf. Availability, Rel. Secur.*, Aug. 2021, pp. 1–11.
[80] H. Becker, V. Hwang, M. J. Kannwischer, B.-Y. Yang, and S.-Y. Yang, "Neon NTT: Faster Dilithium, Kyber, and saber on Cortex-A72 and Apple M1," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2022, pp. 221–244, Nov. 2021.
[81] A. Basso, F. Aydin, D. Dinu, J. Friel, A. Varna, M. Sastry, and S. Ghosh, "Where star wars meets star trek: Saber and Dilithium on the same polynomial multiplier," *Cryptol. ePrint Arch.*, to be published.
[82] D. O. C. Greconici, M. J. Kannwischer, and D. Sprenkels, "Compact Dilithium implementations on Cortex-M3 and Cortex-M4," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, pp. 1–24, Dec. 2020.
[83] V. B. Dang, K. Mohajerani, and K. Gaj, "High-speed hardware architectures and FPGA benchmarking of CRYSTALS-Kyber, NTRU, and saber," *IEEE Trans. Comput.*, vol. 72, no. 2, pp. 306–320, Feb. 2023.
[84] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster interleaved modular multiplication based on Barrett and Montgomery reduction methods," *IEEE Trans. Comput.*, vol. 59, no. 12, pp. 1715–1721, Dec. 2010.
[85] Y. Zhao, R. Xie, G. Xin, and J. Han, "A high-performance domain-specific processor with matrix extension of RISC-V for module-LWE applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2871–2884, Jul. 2022.
[86] D. W. Kim, D. I. Maulana, and W. Jung, "Kyber accelerator on FPGA using energy-efficient LUT-based Barrett reduction," in *Proc. 19th Int. SoC Design Conf. (ISOCC)*, Oct. 2022, pp. 83–84.

[87] L. Wan, F. Zheng, G. Fan, R. Wei, L. Gao, Y. Wang, J. Lin, and J. Dong, "A novel high-performance implementation of CRYSTALS-Kyber with ai accelerator," in *Proc. Eur. Symp. Res. Comput. Secur.* Copenhagen, Denmark: Springer, Sep. 2022, pp. 514–534.

[88] H. Nguyen and L. Tran, "Design of polynomial NTT and INTT accelerator for post-quantum cryptography CRYSTALS-Kyber," *Arabian J. Sci. Eng.*, vol. 48, pp. 1527–1536, Feb. 2022.

[89] P. Sanal, E. Karagoz, H. Seo, R. Azarderakhsh, and M. Mozaffari-Kermani, "Kyber on ARM64: Compact implementations of Kyber on 64-bit arm cortex-a processors," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.* Cham, Switzerland: Springer, Sep. 2021, pp. 424–440.

[90] J. N. Ortiz, F. C. Rodrigues, D. G. Filho, C. Teixeira, J. López, and R. Dahab, "Evaluation of CRYSTALS-Kyber and saber on the ARMv8 architecture," in *Proc. Anais do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, 2022, pp. 372–377.

[91] J. Huang, J. Zhang, H. Zhao, Z. Liu, R. C. C. Cheung, Ç. K. Koç, and D. Chen, "Improved plantard arithmetic for lattice-based cryptography," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2022, pp. 614–636, Aug. 2022.

[92] T. Plantard, "Efficient word size modular arithmetic," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1506–1518, Jul. 2021.

[93] Z. Ye, R. C. C. Cheung, and K. Huang, "PipeNTT: A pipelined number theoretic transform architecture," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 10, pp. 4068–4072, Oct. 2022.

[94] M. Li, J. Tian, X. Hu, and Z. Wang, "Reconfigurable and high-efficiency polynomial multiplication accelerator for CRYSTALS-Kyber," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Dec. 19, 2022, doi: 10.1109/TCAD.2022.3230359.

[95] D. Kundi, Y. Zhang, C. Wang, A. Khalid, M. O'Neill, and W. Liu, "Ultra high-speed polynomial multiplications for lattice-based cryptography on FPGAs," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1993–2005, Oct. 2022.

[96] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "A monolithic hardware implementation of Kyber: Comparing apples to apples in PQC candidates," in *Proc. Int. Conf. Cryptol. Inf. Secur. Latin Amer.* Bogotá, Colombia: Springer, 2021, pp. 108–126.

[97] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, pp. 328–356, Feb. 2021.

[98] P. Nannipieri, S. Di Matteo, L. Zulberti, F. Albicocchi, S. Saponara, and L. Fanucci, "A RISC-V post quantum cryptography instruction set extension for number theoretic transform to speed-up CRYSTALS algorithms," *IEEE Access*, vol. 9, pp. 150798–150808, 2021.

[99] W. Guo, S. Li, and L. Kong, "An efficient implementation of Kyber," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 1562–1566, Mar. 2022.

[100] L. Zhao, J. Zhang, J. Huang, Z. Liu, and G. Hancke, "Efficient implementation of Kyber on mobile devices," in *Proc. IEEE 27th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2021, pp. 506–513.

[101] D. T. Nguyen and K. Gaj, "Fast NEON-based multiplication for lattice-based NIST post-quantum cryptography finalists," in *Proc. Int. Conf. Post-Quantum Cryptogr.* Daejeon, South Korea: Springer, Jul. 2021, pp. 234–254.

[102] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "High-performance area-efficient polynomial ring processor for CRYSTALS-Kyber on FPGAs," *Integration*, vol. 78, pp. 25–35, May 2021.

[103] Z. Liu, H. Seo, S. Sinha Roy, J. Großschädl, H. Kim, and I. Verbauwhede, "Efficient Ring-LWE encryption on 8-bit AVR processors," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Saint-Malo, France: Springer, Sep. 2015, pp. 663–682.

[104] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography," in *Proc. IEEE 28th Symp. Comput. Arithmetic (ARITH)*, Jun. 2021, pp. 94–101.

[105] 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.

[106] L. Ma, X. Wu, and G. Bai, "Parallel polynomial multiplication optimized scheme for CRYSTALS-KYBER post-quantum cryptosystem based on FPGA," in *Proc. Int. Conf. Commun., Inf. Syst. Comput. Eng. (CISCE)*, May 2021, pp. 361–365.

[107] K. Yao, D. Kundi, C. Wang, M. O'Neill, and W. Liu, "Towards CRYSTALS-Kyber: A M-LWE cryptoprocessor with area-time trade-off," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[108] C. Zhang, D. Liu, X. Liu, X. Zou, G. Niu, B. Liu, and Q. Jiang, "Towards efficient hardware implementation of NTT for Kyber on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[109] Y. Huang, M. Huang, Z. Lei, and J. Wu, "A pure hardware implementation of CRYSTALS-KYBER PQC algorithm through resource reuse," *IEICE Electron. Exp.*, vol. 17, no. 17, 2020, Art. no. 20200234.

[110] Z. Chen, Y. Ma, T. Chen, J. Lin, and J. Jing, "Towards efficient Kyber on FPGAs: A processor for vector of polynomials," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 247–252.

[111] L. Botros, M. J. Kannwischer, and P. Schwabe, "Memory-efficient high-speed implementation of Kyber on Cortex-m4," in *Proc. Int. Conf. Cryptol. Africa*. Rabat, Morocco: Springer, Jul. 2019, pp. 209–228.

[112] Y. Kim, J. Song, and S. C. Seo, "Accelerating falcon on ARMv8," *IEEE Access*, vol. 10, pp. 44446–44460, 2022.

[113] W.-K. Lee, R. K. Zhao, R. Steinfeld, A. Sakzad, and S. O. Hwang, "High throughput lattice-based signatures on GPUs: Comparing Falcon and Mitaka," *Cryptol. ePrint Arch.*, to be published.

[114] (2023). *Australian Research Data Commons Nectar Research Cloud System*. [Online]. Available: https://ardc.edu.au/services/

[115] P. Karl, J. Schupp, T. Fritzmann, and G. Sigl, "Post-quantum signatures on RISC-V with hardware acceleration," *ACM Trans. Embedded Comput. Syst.*, to be published.

[116] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017.

[117] E. Karabulut and A. Aysu, "RANTT: A RISC-V architecture extension for the number theoretic transform," in *Proc. 30th Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug. 2020, pp. 26–32.

[118] D. Harvey and J. van der Hoeven, "Polynomial multiplication over finite fields in time $O(n \log n)$," *J. ACM*, vol. 69, no. 2, pp. 1–40, Apr. 2022.

[119] L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Sofia, Bulgaria: Springer, Apr. 2015, pp. 617–640.

[120] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, Jan. 2020.

[121] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Annual Cryptol. Conf.* Santa Barbara, CA, USA: Springer, Aug. 2013, pp. 75–92.

[122] S. Halevi and V. Shoup, "Design and implementation of helib: A homomorphic encryption library," *Cryptol. ePrint Arch.*, to be published.

[123] A. Al Badawi et al., "OpenFHE: Open-source fully homomorphic encryption library," in *Proc. 10th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, Nov. 2022, pp. 53–63.

[124] C. V. Mouchet, J.-P. Bossuat, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, "Lattigo: A multiparty homomorphic encryption library in go," in *Proc. 8th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, 2020, pp. 64–70.

[125] Z. Wang, P. Li, R. Hou, Z. Li, J. Cao, X. Wang, and D. Meng, "HE-booster: An efficient polynomial arithmetic acceleration on GPUs for fully homomorphic encryption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 4, pp. 1067–1081, Apr. 2023.

[126] Z. Zheng, "Encrypted cloud using GPUs," Ph.D. dissertation, KU Leuven, Leuven, Belgium, 2020. [Online]. Available: https://www~

[127] E. Öztürk, Y. Doröz, E. Savas, and B. Sunar, "A custom accelerator for homomorphic encryption applications," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 3–16, Jan. 2017.

[128] W. Wang, Z. Chen, and X. Huang, "Accelerating leveled fully homomorphic encryption using GPU," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 2800–2803.

[129] W. Dai, Y. Doröz, and B. Sunar, "Accelerating NTRU based homomorphic encryption using GPUs," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2014, pp. 1–6.

[130] A. S. Özcan, C. Ayduman, E. R. Türkoglu, and E. Savas, "Homomorphic encryption on GPU," *IEEE Access*, early access, Apr. 7, 2023, doi: 10.1109/ACCESS.2023.3265583.

[131] Ö. Özerk, C. Elgezen, A. C. Mert, E. Öztürk, and E. Savaş, "Efficient number theoretic transform implementation on GPU for homomorphic encryption," *J. Supercomput.*, vol. 78, no. 2, pp. 2840–2872, Feb. 2022.
[132] Y. Su, B. Yang, C. Yang, Z. Yang, and Y. Liu, "A highly unified reconfigurable multicore architecture to speed up NTT/INTT for homomorphic polynomial multiplication," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 8, pp. 993–1006, Aug. 2022.
[133] Y. Doröz, E. Öztürk, and B. Sunar, "Accelerating fully homomorphic encryption in hardware," *IEEE Trans. Comput.*, vol. 64, no. 6, pp. 1509–1521, Jun. 2015.
[134] A. C. Mert, E. Karabulut, E. Öztürk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Comput.*, vol. 71, no. 11, pp. 2829–2843, Nov. 2022.
[135] P. Duong-Ngoc, S. Kwon, D. Yoo, and H. Lee, "Area-efficient number theoretic transform architecture for homomorphic encryption," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 3, pp. 1270–1283, Mar. 2023.
[136] X. Feng and S. Li, "Design of an area-effcient million-bit integer multiplier using double modulus NTT," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 9, pp. 2658–2662, Sep. 2017.
[137] C. Rafferty, M. O'Neill, and N. Hanley, "Evaluation of large integer multiplication methods on hardware," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1369–1382, Aug. 2017.
[138] R. Paludo and L. Sousa, "NTT architecture for a Linux-ready RISC-V fully-homomorphic encryption accelerator," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2669–2682, Jul. 2022.
[139] Y. Su, B. Yang, C. Yang, and S. Zhao, "ReMCA: A reconfigurable multicore architecture for full RNS variant of BFV homomorphic evaluation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2857–2870, Jul. 2022.
[140] A. Al Badawi, Y. Polyakov, K. M. M. Aung, B. Veeravalli, and K. Rohloff, "Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 2, pp. 941–956, Apr. 2021.
[141] X. Cao, C. Moore, M. O'Neill, E. O'Sullivan, and N. Hanley, "Optimised multiplication architectures for accelerating fully homomorphic encryption," *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2794–2806, Sep. 2016.
[142] S. S. Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 387–398.
[143] P. Ravi, R. Poussier, S. Bhasin, and A. Chattopadhyay, "On configurable SCA countermeasures against single trace attacks for the NTT: A performance evaluation study over Kyber and Dilithium on the arm Cortex-m4," in *Proc. Int. Conf. Secur., Privacy, Appl. Cryptogr. Eng.* Kolkata, India: Springer, Dec. 2020, pp. 123–146.
[144] J. Howe, T. Prest, and D. Apon, "SoK: How (not) to design and implement post-quantum cryptography," in *Proc. Cryptographers' Track RSA Conf.* Cham, Switzerland: Springer, May 2021, pp. 444–477.

ARDIANTO SATRIAWAN received the B.S. and M.S. degrees in electrical engineering from Institut Teknologi Bandung (ITB), Bandung, Indonesia, in 2013 and 2015 respectively. He is a member of the Computer Engineering Research Group, School of Electrical Engineering and Informatics, ITB. His research interests include virtual reality, machine learning, computer networks, and information security.

INFALL SYAFALNI received the B.Eng. degree in electrical engineering from Institut Teknologi Bandung (ITB), Bandung, Indonesia, in 2008, the M.Sc. degree in electronic engineering from the University of Science Malaysia (USM), Penang, Malaysia, in 2011, and the Dr.Eng. degree in engineering from the Kyushu Institute of Technology (KIT), Iizuka, Fukuoka, Japan, in 2014. From 2014 to 2015, he held a research position with KIT. From 2015 to 2018, he was an ASIC Engineer with the ASIC Development Group, Logic Research Company Ltd., Fukuoka, Japan. In 2019, he joined ITB, where he is currently an Assistant Professor with the School of Electrical Engineering and Informatics and a Researcher with the University Center of Excellence on Microelectronics. His research interests include logic synthesis, logic design, VLSI design, and efficient circuits and algorithms.

RELLA MARETA (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Institut Teknologi Bandung (ITB), Bandung, Indonesia, in 2011 and 2014, respectively. She is currently pursuing the Ph.D. degree with the Digital Integrated Systems Laboratory, Inha University.

ISA ANSHORI (Member, IEEE) received the B.S. degree in engineering physics from Institut Teknologi Bandung, Indonesia, in 2009, and the M.Eng. degree in materials science and the Ph.D. degree in nanoscience and nanotechnology from the University of Tsukuba, Japan, in 2015 and 2018, respectively. He has been an Assistant Professor with the Department of Biomedical Engineering, School of Electrical Engineering and Informatics, Institut Teknologi Bandung, since 2018. His research interests include bio/chemical sensors, microfluidics, the IoT devices, and lab-on-chip.

WERVYAN SHALANNANDA received the B.S. degree in telecommunications engineering and the M.S. degree in electrical engineering (telematics and telco networks) from the Bandung Institute of Technology, in 2013 and 2015, respectively. He joined the Bandung Institute of Technology, in 2016, as an Academic Assistant and then as a Lecturer, in 2018. His research interests include networked systems and security and artificial intelligence in telecommunications.

ALEAMS BARRA received the B.S. and M.S. degrees in mathematics from Institut Teknologi Bandung (ITB), Bandung, Indonesia, in 1998 and 2002, respectively, and the Ph.D. degree in mathematics from the University of Kentucky, Kentucky, USA, in 2012. He is currently a member of the Algebra Research Group, Faculty of Mathematics and Natural Sciences, ITB.