Accelerating HE Operations from Key Decomposition Technique

Miran Kim[1][0000−0003−3564−6090], Dongwon Lee[2][0000−0002−2156−197X], Jinyeong Seo[2][0000−0001−9080−5272], and Yongsoo Song[2][0000−0002−0496−9789]

[1] Department of Mathematics & Research Institute for Convergence of Basic Science & Hanyang Institute of Bioscience and Biotechnology,
Hanyang University, Republic of Korea
miran@hanyang.ac.kr
[2] Seoul National University, Republic of Korea
{dongwonlee95, jinyeong.seo, y.song}@snu.ac.kr

Abstract. Lattice-based homomorphic encryption (HE) schemes are based on the noisy encryption technique, where plaintexts are masked with some random noise for security. Recent advanced HE schemes rely on a decomposition technique to manage the growth of noise, which involves a conversion of a ciphertext entry into a short vector followed by multiplication with an evaluation key. Prior to this work, the decomposition procedure turns out to be the most time-consuming part, as it requires discrete Fourier transforms (DFTs) over the base ring for efficient polynomial arithmetic. In this paper, an expensive decomposition operation over a large modulus is replaced with relatively cheap operations over a ring of integers with a small bound. Notably, the cost of DFTs is reduced from quadratic to linear with the level of a ciphertext without any extra noise growth. We demonstrate the implication of our approach by applying it to the key-switching procedure. Our experiments show that the new key-switching method achieves a speedup of 1.2–2.3 or 2.1–3.3 times over the previous method, when the dimension of a base ring is 2^{15} or 2^{16}, respectively.

Keywords: Homomorphic Encryption · Gadget Decomposition · External Product.

1 Introduction

Homomorphic Encryption (HE) is a cryptosystem that allows us to compute on encrypted data without decrypting them. Since Gentry's pioneering work [17], a number of HE schemes have been proposed based on lattice cryptography. In particular, the current best-performing HE schemes rely on the hardness of the Learning with Errors (LWE) problem or its ring variant (RLWE) [28, 29], which enables practically usable implementations using algebraic properties.

In (R)LWE-based HE schemes, a small noise is introduced when generating a public key or encrypting a plaintext to ensure security. This noise grows with homomorphic computation, so ciphertexts natively contain some errors which should be kept small enough for correct decryption. To reduce the noise growth derived from nonlinear HE operations, recent advanced HE schemes such as BGV [4], B/FV [3, 16], GSW [19], FHEW/TFHE [14, 12] and CKKS [9] commonly use a decomposition technique that transforms a ciphertext component into a vector of small entries before multiplying it to a public (evaluation) key. Such a combined operation of decomposition and multiplication (which will be referred to as the *external product* in this paper) has been a core building block of HE constructions.

Gentry et al. [18] proposed several optimization techniques, many of which are still used extensively in HE implementations. For example, they introduced polynomial representation methods based on a Residue Number System (RNS) to support efficient polynomial arithmetic over a multi-precision modulus. Since then, Bajard et al. [1] and the subsequent studies [8, 20] presented "full RNS" based constructions of B/FV and CKKS where all computations including gadget decomposition can be performed in an RNS representation. It enables a significant enhancement in performance, and most state-of-the-art HE libraries adopt RNS-friendly HE algorithms for efficiency.

More recently, much progress has been made towards improving the efficiency of bootstrapping technique [7, 5, 26, 2, 27] and investigating HE-compatible approximations of non-polynomial functions [10, 11]. But unfortunately, there has been no remarkable theoretical advance in fundamental HE algorithms in the past few years. Instead, recent studies and projects such as the Data Protection in Virtual Environments (DPRIVE) program [13] are aiming at optimizing implementations, for example, parallelization using GPU [23] and hardware accelerator [30, 31, 25].

Our Contribution. Let $R = \mathbb{Z}[X]/(X^N + 1)$ be a ring of integers for a power-of-two integer N and $R_Q = \mathbb{Z}_Q[X]/(X^N+1)$ be its residue ring modulo an integer Q. The external product is a dyadic operation that takes as its input a polynomial $a \in R_Q$ and a vector $\mathbf{u} = (u_i)_{0\le i<d} \in R_Q^d$.[3] More precisely, it first transforms the input a into its *gadget decomposition*, say $(b_i)_{0\le i<d}$, which is a vector of small elements of R such that $a = \sum_i b_i \cdot g_i \pmod Q$ where $(g_i)_{0\le i<d}$ is a fixed gadget basis over R_Q. And then, it returns the linear combination $\sum_i b_i \cdot u_i \in R_Q$ of u_i's with coefficients b_i.

Our motivation stems from the observation that the previous external product method involves a preprocessing procedure before multiplications between elements of R and R_Q. To be specific, the elements b_i's are first converted to integers modulo Q (denoted $[b_i]_Q$), so that the products $b_i \cdot u_i = [b_i]_Q \cdot u_i$ can be performed using modular multiplications over the modulus Q. As a result, the performance of the external product mainly depends on the conversion process $b_i \mapsto [b_i]_Q$, which is followed by discrete Fourier transforms (DFTs) over the ring R_Q for efficient polynomial arithmetic. Note that these operations are performed over the multi-precision modulus Q, so the complexity of the external product is mainly dependent on the precision.

In this paper, we present a new method to compute the external product more efficiently. A distinguishing feature of our approach is that $\mathbf{u} = (u_i)_{0\le i<d}$ is represented in a decomposed form, say $(v_{i,j})_{0\le i,j<d}$, when it is given as an input to our algorithm. That is, we have $u_i = \sum_j v_{i,j} \cdot \tilde{g}_j \pmod Q$ where $(\tilde{g}_j)_{0\le j<d}$ is another fixed gadget basis over R_Q. We observe that the external product between a and $\mathbf{u}$ can be expressed as a linear combination of the gadget basis:

$$\sum_{0\le j<d} \left(\sum_{0\le i<d} b_i \cdot v_{i,j} \right) \cdot \tilde{g}_j.$$

Hence, our algorithm mainly aims to compute $\sum_i b_i \cdot v_{i,j} \in R$ for all j, which only requires arithmetic operations between small elements of R without any modular reduction. We also point out that $\sum_i b_i \cdot v_{i,j}$ has an upper bound determined by the gadget decomposition and it is significantly smaller than the modulus Q. Therefore, this process can be implemented using DFTs on the element b_i over the polynomial ring with small coefficients rather than R_Q, and therefore the cost of DFTs is independent of the modulus Q. Notably, it is reduced from $O(\ell^2)$ to $O(\ell)$, where ℓ denotes the level of the input polynomial. As a result, the new external product method shows a significant speedup over the previous method.

Our algorithm has a wide range of applications since the external product is widely used in homomorphic operations such as the key-switching technique. For example, the external product between a ciphertext entry and an evaluation key (a gadget encryption of an old key under a new key) can be used to generate a new ciphertext that encrypts (approximately) the same plaintext under the new key. On the other hand, it is worth noting that the proposed method is to compute the same formula in a different way from the previous method, so it does not bring any extra noise growth during the computation. Furthermore, our method offers another advantage in flexibility such that it is compatible with commonly used HE techniques such as hoisting [21] and lazy key-switching [24].

Finally, we implement our external product algorithm and demonstrate its effectiveness through concrete performance. Our experimental results show that the key-switching method based on the new external product method achieves a speedup of 1.2–2.3 and 2.1–3.3 times over the previous method when the dimension of a base ring is 2^{15} and 2^{16}, respectively. For example, with the ring dimension $N = 2^{16}$, the previous key-switching and our key-switching take about 2.7 seconds and 0.8 seconds, respectively.

[3] Here we assume for simplicity that two inputs have the same modulus, but a more general case will be discussed later.

2 Preliminaries

Notations. Let N be a power of two and Q be an integer. We denote by $K = \mathbb{Q}[X]/(X^N + 1)$ the $(2N)$-th cyclotomic field, $R = \mathbb{Z}[X]/(X^N + 1)$ the ring of integers of K, and $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$ the residue ring of R modulo Q. We identify an element $a = \sum_{0 \le i < N} a_i \cdot X^i \in R_Q$ with the vector of its coefficients $(a_0, \ldots, a_{N-1}) \in \mathbb{Z}_Q^N$. We use $\mathbb{Z} \cap (-Q/2, Q/2]$ as a representative of $\mathbb{Z}_Q$ for an integer q, and denote by $[a]_Q$ the reduction of an integer a modulo Q. For $a \in R$, we define $\|a\|_\infty$ as the ℓ_∞-norm of its coefficient vector.

For a real number r, $\lfloor r \rceil$ denotes the nearest integer to r, rounding upwards in case of a tie. For a finite set S, we use $x \leftarrow S$ to denote the sampling x according to the uniform distribution over S. For $\sigma > 0$, we denote by D_σ a distribution over R sampling N coefficients independently from the discrete Gaussian distribution of variance σ^2.

2.1 Ring Learning with Errors

Let χ be a distribution over R and $\sigma > 0$ a real. The ring learning with errors (RLWE) assumption with respect to the parameter (N, Q, χ, σ) is that given polynomially many samples of either (a, b) or $(a, as + e)$, where $a, b \leftarrow R_Q$, $s \leftarrow \chi$, $e \leftarrow D_\sigma$, it is computationally hard to distinguish which is the case. The lattice-based HE schemes such as B/FV [3, 16] and CKKS [9] rely on the security on the RLWE assumption.

2.2 Gadget Decomposition and External Product

We give an overview of 'gadget toolkit' that is commonly used in lattice-based HE cryptosystems for noise reduction.

Definition 1 (Gadget Decomposition). *For a modulus Q, a function $h : R_Q \to R^d$ is called a gadget decomposition if there exists a fixed vector $\mathbf{g} = (g_0, g_1, \ldots, g_{d-1}) \in R_Q^d$ and a real $B > 0$ such that the following holds for any $a \in R_Q$ and its decomposition $\mathbf{b} = (b_0, b_1, \ldots, b_{d-1}) \leftarrow h(a)$:*

$$\sum_{0 \le i < d} b_i \cdot g_i = a \pmod{Q} \qquad \text{and} \qquad \|\mathbf{b}\|_\infty \le B.$$

We call $\mathbf{g}$ a *gadget vector* and $B > 0$ a bound of h.

Definition 2 (External Product). *Let $\tilde{Q} = P \cdot Q$. The external product is a binary operation $\boxdot : R_Q \times R_{\tilde{Q}}^d \to R_Q$ defined as follows: for $a \in R_Q$ and $\mathbf{u} = (u_0, \ldots, u_{d-1}) \in R_{\tilde{Q}}^d$, compute $\mathbf{b} = (b_0, \ldots, b_{d-1}) \leftarrow h(a)$ and $\tilde{c} = \sum_{0 \le i < d} b_i \cdot u_i \pmod{\tilde{Q}}$, and return $c = \left\lfloor \frac{1}{P} \cdot \tilde{c} \right\rceil \pmod{Q}$.*

By a slight abuse of notation, we write $a \boxdot \mathbf{U} = (a \boxdot \mathbf{u}_0, a \boxdot \mathbf{u}_1) \in R_Q^2$ for any $a \in R_Q$ and $\mathbf{U} = [\mathbf{u}_0 | \mathbf{u}_1] \in R_{\tilde{Q}}^{d \times 2}$.

2.3 Homomorphic Encryption and Key-switching

Homomorphic encryption is a cryptosystem that allows computation over encrypted data without decrypting them. We provide a description of the CKKS scheme [9] as an instantiation to explain how gadget decomposition and external product techniques are used in HE systems.

• $\underline{\texttt{CKKS.Setup}(1^\lambda)}$: For given a security paramter λ, choose an RLWE dimension N, a ciphertext modulus Q, a special modulus P, a key distribution χ, and an error parameter $\sigma > 0$. Output the public parameter $\texttt{pp} = (N, Q, P, \chi, \sigma)$. We write $\tilde{Q} = P \cdot Q$.

• $\underline{\texttt{CKKS.KeyGen}(\texttt{pp})}$: Given a public parameter $\texttt{pp}$, generate a secret key, a public key, a relinearization key, and an automorphism key as follows.

- Sample $s \leftarrow \chi$ and return the secret key $\mathsf{sk} \leftarrow s$.
- Sample $p_1 \leftarrow R_{\tilde{Q}}$ and $e_p \leftarrow D_\sigma$, and return the public key $\mathsf{pk} \leftarrow \mathbf{p} = (p_0, p_1) \in R_{\tilde{Q}}^2$ where $p_0 = -p_1 s + e_p \pmod{\tilde{Q}}$.
- Sample $\mathbf{r}_1 \leftarrow R_{\tilde{Q}}^d$ and $\mathbf{e}_r \leftarrow D_\sigma^d$. Generate the relinearization key as $\mathsf{rlk} \leftarrow \mathbf{R} = [\mathbf{r}_0 | \mathbf{r}_1] \in R_{\tilde{Q}}^{d \times 2}$ where $\mathbf{r}_0 = -s \cdot \mathbf{r}_1 + Ps^2 \cdot \mathbf{g} + \mathbf{e}_r \pmod{\tilde{Q}}$.
- For an automorphism ϕ over K, sample $\mathbf{t}_1 \leftarrow R_{\tilde{Q}}^d$ and $\mathbf{e}_t \leftarrow D_\sigma^d$. Generate the automorphism key as $\mathsf{atk} \leftarrow \mathbf{T} = [\mathbf{t}_0 | \mathbf{t}_1] \in R_{\tilde{Q}}^{d \times 2}$ where $\mathbf{t}_0 = -s \cdot \mathbf{t}_1 + P\phi(s) \cdot \mathbf{g} + \mathbf{e}_t \pmod{\tilde{Q}}$.

• $\underline{\mathtt{CKKS.Enc}(\mathsf{pk}; \mu)}$: Given a public key $\mathsf{pk} = \mathbf{p}$ and a plaintext $\mu \in R^4$, output a ciphertext $\mathsf{ct} = \left\lfloor \frac{1}{P} \cdot (v \cdot \mathbf{p} + \mathbf{e}) \right\rceil + (\mu, 0) \pmod{Q}$ where $v \leftarrow \chi$ and $\mathbf{e} \leftarrow D_\sigma^2$.

• $\underline{\mathtt{CKKS.Dec}(s; \mathsf{ct})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_Q^2$ and a secret key $s \in R$, output the plaintext $\mu = c_0 + s \cdot c_1 \pmod{Q}$.

• $\underline{\mathtt{CKKS.Add}(\mathsf{ct}, \mathsf{ct}')}$: Given two ciphertexts $\mathsf{ct}, \mathsf{ct}' \in R_Q^2$, output the ciphertext $\mathsf{ct}_{add} = \mathsf{ct} + \mathsf{ct}' \in R_Q^2$.

• $\underline{\mathtt{CKKS.Mult}(\mathsf{rlk}; \mathsf{ct}, \mathsf{ct}')}$: Given two ciphertexts $\mathsf{ct} = (c_0, c_1)$, $\mathsf{ct}' = (c_0', c_1') \in R_Q^2$ and a relinearization key $\mathsf{rlk} = \mathbf{R}$, let $d_0 = c_0 c_0' \pmod{Q}$, $d_1 = (c_0 c_1' + c_0' c_1) \pmod{Q}$, and $d_2 = c_1 c_1' \pmod{Q}$. Output a ciphertext $\mathsf{ct}_{mul} = (d_0, d_1) + d_2 \boxdot \mathbf{R} \in R_Q^2$.

• $\underline{\mathtt{CKKS.Auto}(\mathsf{atk}, \phi; \mathsf{ct})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_Q^2$, an automorphism key $\mathsf{atk} = \mathbf{U}$, and an automorphism ϕ on R, output a ciphertext $\mathsf{ct}_{aut} = (\phi(c_0), 0) + \phi(c_1) \boxdot \mathbf{U} \in R_Q^2$.

A *gadget encryption* of a plaintext $\mu \in R$ under a secret key s is defined as $\mathbf{U} = [\mathbf{u}_0 | \mathbf{u}_1] \in R_{\tilde{Q}}^{d \times 2}$ where $\mathbf{u}_1 \leftarrow R_{\tilde{Q}}^d$, $\mathbf{e} \leftarrow D_\sigma^d$, and $\mathbf{u}_0 = -s \cdot \mathbf{u}_1 + P\mu \cdot \mathbf{g} + \mathbf{e} \pmod{\tilde{Q}}$. For example, the relinearization and automorphism keys are gadget encryptions of s^2 and $\phi(s)$. The external product can be used for multiplication of an arbitrary element in R_Q and a gadget encryption: for any $a \in R_Q$ and a gadget encryption $\mathbf{U} = [\mathbf{u}_0 | \mathbf{u}_1]$ of a plaintext $\mu \in R$ under a secret s, their external product $(c_0, c_1) \leftarrow a \boxdot \mathbf{U}$ satisfies that

$$c_0 + c_1 \cdot s \approx a \boxdot (\mathbf{u}_0 + s \cdot \mathbf{u}_1) \approx a \boxdot (P\mu \cdot \mathbf{g}) = a \cdot \mu \pmod{Q}.$$

The *key-switching* procedure is a major use-case of the external product. For given a ciphertext component a which is decryptable by a secret s', we can perform the external product with a gadget encryption $\mathbf{U}$ of s' under s (which is often referred as a key-switching key from s' to s) to transform it into an RLWE ciphertext under s while preserving the underlying plaintext. In other words, the resulting ciphertext $(c_0, c_1) \leftarrow a \boxdot \mathbf{U}$ satisfies that $c_0 + c_1 \cdot s \approx a \cdot s' \pmod{Q}$. For example, homomorphic multiplication and automorphism algorithms of CKKS involve the key-switching procedures from s^2 and $\phi(s)$ to s, respectively.

2.4 Polynomial Representations

The *Residue Number System* (RNS) enables representing a large integer as a tuple of small integers. Let $Q = q_0 \cdots q_{\ell-1}$ where q_k's are pairwise coprime integers. Then, we get an isomorphism $R_Q \to R_{q_0} \times \cdots \times R_{q_{\ell-1}}$, $a \mapsto ([a]_{q_0}, \ldots, [a]_{q_{\ell-1}})$ from the Chinese remainder theorem. The image $([a]_{q_0}, \ldots, [a]_{q_{\ell-1}})$ of an element $a \in R_Q$ is called the RNS representation of a. The main advantage of the RNS representation is that one can instantiate R_Q with smaller rings R_{q_k}'s. There have been several studies [1, 8, 20] to exploit the RNS representation for optimization of HE schemes by avoiding multi-precision arithmetic of a base ring.

[4] The original CKKS scheme enables to encode a vector of complex numbers to a plaintext in R. For the sake of brevity, we assume that an input of the encryption function is given as a plaintext polynomial.

We also define a discrete Fourier transformation over $\mathbb{Z}_{q_k}$, called *Number Theoretic Transform (NTT)*, for efficient polynomial arithmetic over R_{q_k}. Suppose that q_k is a prime integer such that $q_k = 1 \pmod{2N}$. Then, there exists a primitive $(2N)$-th root of unity ξ_k modulo q_k. It is easy to show that

$$a(X) \mapsto \left(a(\xi_k^i)\right)_{i \in \mathbb{Z}_{2N}^\times}$$

is an isomorphism from R_{q_k} to $\mathbb{Z}_{q_k}^N$. We call $\left(a(\xi_k^i)\right)_{i \in \mathbb{Z}_{2N}^\times}$ the NTT representation of $a \in R_{q_k}$. More generally, when Q is a product of distinct primes $q_0, \ldots, q_{\ell-1}$ such that $q_k = 1 \pmod{2N}$, we can combine NTTs over different moduli q_k to convert the RNS representation of $a \in R_Q$ between its coefficient and NTT forms. We will say that a ciphertext is in the coefficient (or NTT) form if its components are represented in the coefficient (or NTT, respectively) form.

3 A New External Product Method

Let $h : R_Q \to R^d$ be a gadget decomposition corresponding to a gadget vector $\mathbf{g} \in R_Q^d$. Recall that an external product of $a \in R_Q$ and $\mathbf{u} = (u_i)_{0 \le i < d} \in R_{\tilde{Q}}^d$ involves the computation of the gadget decomposition $\mathbf{b} = (b_i)_{0 \le i < d}$ of a, followed by the linear combination of $u_0, \ldots, u_{d-1} \in R_{\tilde{Q}}$ with coefficients $b_0, \ldots, b_{d-1} \in R$. Note that this operation is well-defined since $R_{\tilde{Q}}$ is an R-module.

In previous studies [8, 20], the linear combination $\sum_i b_i \cdot u_i$ is obtained by converting $\mathbf{b} \in R^d$ into $[\mathbf{b}]_{\tilde{Q}} \in R_{\tilde{Q}}^d$ and computing the inner product between $[\mathbf{b}]_{\tilde{Q}}$ and $\mathbf{u}$ over $R_{\tilde{Q}}$. The first step is particularly expensive since each conversion $b_i \mapsto [b_i]_{\tilde{Q}}$ involves NTT operations modulo the prime factors of $\tilde{Q}$.

In this section, we present a new approach to perform the external product operation more efficiently. In a nutshell, our approach employs gadget decomposition to represent the entries u_i as small elements in the ring of integers R. It enables us to express the external product using low-precision arithmetic over R. As a result, we can achieve performance improvements since the NTT operations over $R_{\tilde{Q}}$ are replaced with DFTs over R where the precision is determined by the (small) bounds of gadget decompositions rather than the (large) modulus $\tilde{Q}$.

3.1 Main Idea

Intuitively, the gadget decomposition can be understood as an operation to represent an arbitrary element of a residue ring as a linear combination of a fixed basis with small coefficients. Meanwhile, the external product involves a linear combination of arbitrary elements $u_i \in R_{\tilde{Q}}$ with small coefficients $b_i \in R$. The main observation is that, if u_i's are given as linear combinations with small coefficients, say $v_{i,j}$, then $\sum_i b_i \cdot u_i$ can be also expressed as a linear combination of the same basis where its coefficients can be obtained from $v_{i,j}$ and b_i.

Specifically, we introduce another gadget decomposition $\tilde{\mathbf{g}} = (\tilde{g}_0, \ldots, \tilde{g}_{\tilde{d}-1}) \in R_{\tilde{Q}}^{\tilde{d}}$ over $R_{\tilde{Q}}$ with a gadget vector $\tilde{h} : R_{\tilde{Q}} \to R^{\tilde{d}}$. For given $\mathbf{u} = (u_i)_{0 \le i < d} \in R_{\tilde{Q}}^d$, we decompose its entries via $\tilde{h}$ and get $\tilde{h}(u_i) = (v_{i,0}, \ldots, v_{i,\tilde{d}-1}) \in R^{\tilde{d}}$ for $0 \le i < d$. We write $\mathbf{V} := [v_{i,j}]_{0 \le i < d, 0 \le j < \tilde{d}}$ and denote its j-th column vector by $\mathbf{v}_j$ for $0 \le j < \tilde{d}$. Then, $\mathbf{V} = [\mathbf{v}_0 | \ldots | \mathbf{v}_{\tilde{d}-1}] \in R^{d \times \tilde{d}}$ can be viewed as gadget decomposition of $\mathbf{u}$ satisfying $\mathbf{u} = \sum_{0 \le j < \tilde{d}} \mathbf{v}_j \cdot \tilde{g}_j \pmod{\tilde{Q}}$. We note that $\mathbf{u}$ is usually a public material (such as a key-switching key generated by the key owner) in HE construction, so its gadget decomposition $\mathbf{V}$ can be pre-computed and reused during homomorphic evaluation.

Suppose that we are given $a \in R_Q$ and wish to compute the external product $a \boxdot \mathbf{u}$. Then, given the decomposition $\mathbf{b} = (b_i)_{0 \le i < d} \leftarrow h(a)$, the linear combination $\sum_i b_i \cdot u_i$ can be expressed using the gadget

decomposition $\mathbf{V}$ of $\mathbf{u}$ as follows:

$$\begin{aligned}\sum_{0\le i<d} b_i \cdot u_i &= \begin{bmatrix} b_0 \dots b_{d-1} \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ \vdots \\ u_{d-1} \end{bmatrix} \\ &= \begin{bmatrix} b_0 \dots b_{d-1} \end{bmatrix} \cdot \begin{bmatrix} v_{0,0} & \cdots & v_{0,\tilde{d}-1} \\ \vdots & \ddots & \vdots \\ v_{d-1,0} & \cdots & v_{d-1,\tilde{d}-1} \end{bmatrix} \cdot \begin{bmatrix} \tilde{g}_0 \\ \vdots \\ \tilde{g}_{\tilde{d}-1} \end{bmatrix} \\ &= (\mathbf{b}^\top \cdot \mathbf{V}) \cdot \tilde{\mathbf{g}} = \sum_{0\le j<\tilde{d}} \langle \mathbf{b}, \mathbf{v}_j \rangle \cdot \tilde{g}_j \pmod{\tilde{Q}}. \end{aligned} \tag{1}$$

Note that $\mathbf{b}$ and $\mathbf{v}_0, \dots, \mathbf{v}_{\tilde{d}-1}$ are vectors in R^d, so the inner products $\langle \mathbf{b}, \mathbf{v}_j \rangle$ are defined over the ring R. As a result, $\sum_i b_i \cdot u_i$ is a linear combination of $\tilde{g}_0, \dots, \tilde{g}_{\tilde{d}-1}$ with coefficients $\langle \mathbf{b}, \mathbf{v}_0 \rangle, \dots, \langle \mathbf{b}, \mathbf{v}_{\tilde{d}-1} \rangle$.

Finally, we need to carry out multiplications between $\langle \mathbf{b}, \mathbf{v}_j \rangle$ and $\tilde{g}_j$ over $R_{\tilde{Q}}$ for $0 \le j < \tilde{d}$ to complete the computation. Fortunately, this step can be very cheap when the gadget decomposition $\tilde{h}$ is chosen properly (which will be explained later in the next section). We also point out that the proposed method is purely an algorithmic optimization technique which outputs exactly the same result as before, thereby bringing no extra noise growth.

3.2 Representation and Arithmetic of Integral Polynomials

As discussed above, our external product method involves arithmetic operations over the ring R. Hence, it is necessary to estimate an upper bound of a given computational task to guarantee the correctness and efficiency of its implementation.

Let B and $\tilde{B}$ be upper bounds of two gadget decompositions h and $\tilde{h}$, respectively. In the new external product method, the inner products $\langle \mathbf{b}, \mathbf{v}_j \rangle$ are computed over the ring R, and these are bounded by

$$\left\| \langle \mathbf{b}, \mathbf{v}_j \rangle \right\|_\infty \le dN \cdot \left\| \mathbf{b} \right\|_\infty \cdot \left\| \mathbf{v}_j \right\|_\infty \le dN \cdot B\tilde{B}.$$

On the other hand, an element of R can be represented as the vector of its coefficients in $\mathbb{Z}^N$, but we use the DFT algorithm and its inverse over R to convert the coefficient vector into the DFT form and vice versa for efficient arithmetic operations. More specifically, we instantiate the polynomial ring R as $R_{B'}$ for some integer $B' > 2dN \cdot B\tilde{B}$ so that the reduction modulo B' does not occur during the computation of $\langle \mathbf{b}, \mathbf{v}_j \rangle \in R$ over $R_{B'}$. In our implementation, we set $B' = p_0 p_1 \dots p_{r'-1}$ as a product of distinct word-size primes such that $p_j = 1 \pmod{2N}$ for a possibly minimal integer r', and use the isomorphism $R_{B'} \cong \prod_{0\le j<r'} R_{p_j}$ to represent the elements in $R_{B'}$ in the RNS form with moduli $p_0, \dots, p_{r'-1}$. Then, the DFT over $R_{B'}$ can be efficiently implemented via small NTTs over $R_{p_0}, \dots, R_{p_{r'-1}}$. Since $B' \approx 2dN \cdot B\tilde{B}$ is significantly smaller than $\tilde{Q}$, the unit complexity of the DFT algorithm over $R_{B'}$ in our external product method is much cheaper than that of the previous NTT operations over $R_{\tilde{Q}}$. A more detailed performance analysis will be provided in the next section.

4 Application to Key Switching

The external product operation is widely used in HE-based constructions such as the GSW scheme [19] and multi-key HE scheme [6]. In particular, as noted in Sec. 2.3, the key-switching operation in HE cryptosystems is a major application of the external product operation. More specifically, the key switching mechanism is extensively used in nonlinear homomorphic operations such as multiplication or automorphism; however, it is the main bottleneck affecting the performance of such homomorphic operations. For instance, the key-switching spends 86-93% of its time performing homomorphic multiplication (the detailed timing results will be provided later in Sec. 5.2). Therefore, improving the key-switching operation can lead to a substantial efficiency enhancement of HE schemes.

In the following, we will focus on the key-switching operation based on our new external product method. We first introduce a practical gadget decomposition that is compatible with the proposed key-switching method and explain how it can lead to performance improvements. We also provide a complexity analysis of the previous and our key-switching methods. We will count the number of unit NTT operations and unit Hadamard operations where NTT operation on R_q and Hadamard product on $\mathbb{Z}_q^N$ are set as unit NTT operation and unit Hadamard product for a word-sized prime integer q, respectively.

4.1 RNS-based Gadget Decomposition

Since Gentry et al. [18] presented an RNS-based representation of polynomials, it has been extensively used in HE implementations to optimize arithmetic operations of a base polynomial ring. Moreover, recent studies [1, 8, 20, 22] showed that it enables to construct HE schemes in a full-RNS manner where all computations are performed in RNS representation without any high-precision arithmetic. We provide a brief overview of RNS-friendly gadget decomposition methods presented in [22].

Throughout this section, let $Q = q_0 \cdots q_{\ell-1}$ for a set of distinct word-sized primes $\mathcal{B} = \{q_0, \ldots, q_{\ell-1}\}$. We partition this RNS base $\mathcal{B}$ into $\mathcal{B}_j = \{q_k : k \in I_j\}$ for $0 \le j < d$ where $I_j \subseteq I = \{0, 1, \ldots, \ell-1\}$ denotes the j-th index set. Note that the *digits* $D_j = \prod_{k \in I_j} q_k$ are pairwise coprime integers such that $Q = D_0 \ldots D_{d-1}$. We call $|I_j|$ the *length* of a digit D_j.

We define the RNS-based *digit decomposition* $h : R_Q \to R^d$ as $a \mapsto \mathbf{b} = (b_0, \ldots, b_{d-1})$ for $b_j = [a]_{D_j}$. We also write $\hat{D}_j = Q/D_j$ for $0 \le j < d$ and let $\mathbf{g} = (g_0, \ldots, g_{d-1}) \in R_Q^d$ for $g_j = [\hat{D}_j^{-1}]_{D_j} \cdot \hat{D}_j$. Then, it is easy to show that h is a gadget decomposition corresponding to $\mathbf{g}$ with a bound $\|\mathbf{b}\|_\infty \le B = \frac{1}{2}\max_{0 \le j < d}\{D_j\}$ from the property that $g_j = 1 \pmod{D_j}$ and $g_j = 0 \pmod{D_{j'}}$ for $j' \ne j$.

The digit decomposition can be computed in an RNS representation using the *base conversion* algorithm [1, 20], which can extend the RNS basis of a polynomial. Let $\hat{q}_k = D_j/q_k$ and $q_k^* = [\hat{q}_k^{-1}]_{\hat{q}_k}$ for $0 \le j < d$ and $k \in I_j$. Suppose that we are given the RNS representation $(a_k = [a]_{q_k})_{0 \le k < \ell}$ of an element $a \in R_Q$. Then, we get $a = \sum_{k \in I_j} [a_k \cdot q_k^*]_{q_k} \cdot \hat{q}_k \pmod{D_j}$ and obtain the following equations over R:

$$b_j = \sum_{k \in I_j} [a_k \cdot q_k^*]_{q_k} \cdot \hat{q}_k - D_j \cdot z_j, \tag{2}$$

where

$$z_j = \Big\lfloor \frac{1}{D_j} \cdot \Big(\sum_{k \in I_j} [a_k \cdot q_k^*]_{q_k} \cdot \hat{q}_k \Big) \Big\rceil = \Big\lfloor \sum_{k \in I_j} \frac{[a_k \cdot q_k^*]_{q_k}}{q_k} \Big\rceil. \tag{3}$$

Based on these formulae, we can represent the digit decomposition in a full-RNS manner as follows. Suppose that we are given $a_k = [a]_{q_k}$ for $k \in I_j$ and wish to compute $b_j = [a]_{D_j}$ modulo Q' for $0 \le j < d$. Then, for each prime factor q' of Q', we have

$$b_j = \Big(\sum_{k \in I_j} [a_k \cdot q_k^*]_{q_k} \cdot [\hat{q}_k]_{q'} \Big) - [D_j]_{q'} \cdot z_j \pmod{q'}, \tag{4}$$

where the constants such as $[\hat{q}_k]_{q'}$ and $[D_j]_{q'}$ are pre-computable independently from the input a. Hence, the RNS representation of b_j with respect to the modulus Q' is obtained by computing Eq. (4) for all prime factors q' of Q'.

4.2 Previous Key-switching Method

We first review the concrete details of the previous key-switching method. We denote by $Q = q_0 \cdots q_{\ell-1}$ and $P = q_\ell \cdots q_{\tilde{\ell}-1}$ the ciphertext modulus and special modulus, respectively, for distinct word-sized primes $q_0, \ldots, q_{\tilde{\ell}-1}$, and let $\tilde{Q} = P \cdot Q = q_0 \cdots q_{\tilde{\ell}-1}$. We choose a partition of the RNS base $\mathcal{B} = \{q_0, \ldots, q_{\ell-1}\}$ and set digits D_j as in Sec. 4.1 so that the digit decomposition $h : R_Q \to R^d$ over R_Q is defined as $h : a \mapsto \mathbf{b} = (b_0, \ldots, b_{d-1})$ for $b_j = [a]_{D_j}$. We denote by $r_j = |I_j|$ the length of a digit D_j. We assume that all computations are performed in RNS representation even if not mentioned explicitly.

Previous key-switching operation

Input: A polynomial $a \in R_Q$ and a key-switching key $\mathbf{U} = [\mathbf{u}_0|\mathbf{u}_1] \in R_{\tilde{Q}}^{d\times 2}$.
Step 1 (Decomposition): Generate the RNS representation of $\mathbf{b} = (b_0, \ldots, b_{d-1}) \leftarrow h(a)$ modulo $\tilde{Q}$. Each $[b_j]_{\tilde{Q}}$ can be computed using Eq. (4) for all prime factors $q_0, \ldots, q_{\tilde{\ell}-1}$ of $\tilde{Q}$.
Step 2 (Inner product): Compute $\tilde{c}_0 = \langle \mathbf{b}, \mathbf{u}_0 \rangle \pmod{\tilde{Q}}$ and $\tilde{c}_1 = \langle \mathbf{b}, \mathbf{u}_1 \rangle \pmod{\tilde{Q}}$
Step 3 (Mod reduction): Compute $c'_0 = \left\lfloor \frac{1}{P}\tilde{c}_0 \right\rceil \pmod{Q}$ and $c'_1 = \left\lfloor \frac{1}{P}\tilde{c}_1 \right\rceil \pmod{Q}$.
Output: An RLWE ciphertext $(c'_0, c'_1) \in R_Q^2$.

The previous key-switching procedure consists of three steps – gadget decomposition, inner product, and modulus reduction. The first step aims to compute the gadget decomposition $h(a)$, but more precisely, its RNS representation over the modulus $\tilde{Q} = q_0 \cdots q_{\tilde{\ell}-1}$. Suppose that $a \in R_Q$ is given in the coefficient form. Then, we compute $b_j = [a]_{D_j}$ using Equations (3) and (4) for all prime factors $q' = q_0, q_1, \ldots, q_{\tilde{\ell}-1}$, and transform it into the NTT form over $R_{\tilde{Q}}$ for an efficient multiplication in the next step. In the second step, we perform two inner products $\tilde{c}_i = \langle \mathbf{b}, \mathbf{u}_i \rangle \pmod{\tilde{Q}}$ for $i = 0, 1$. We assume that the key-switching key $\mathbf{U} = [\mathbf{u}_0|\mathbf{u}_1] \in R_{\tilde{Q}}^{d\times 2}$ is given in the NTT form over $R_{\tilde{Q}}$ for efficiency. Then, each inner product between $\mathbf{b}$ and $\mathbf{u}_i$ is simply written as N inner products over $\mathbb{Z}_{\tilde{Q}}^d \cong \prod_{0\le k<\tilde{\ell}} \mathbb{Z}_{q_k}^d$ since both are given in their NTT forms. At the end of this step, $\tilde{c}_0$ and $\tilde{c}_1$ are converted into the coefficient form for the next step. Finally, we perform the modulus reduction operation to scale the ciphertexts back down to Q, i.e., compute $c'_i = \lfloor \tilde{c}_i/P \rceil = (\tilde{c}_i - [\tilde{c}_i]_P)/P$ for $i = 0, 1$. This step involves a base conversion to obtain the RNS representation of $[\tilde{c}_i]_P$ modulo Q.

Complexity analysis. For the sake of brevity, we assume that all the digit lengths r_j are the same and denoted by r, so we get $\ell = dr$. We first count the number of unit NTT operations in the previous key-switching operation. In the first step, we convert the decomposed element $[b_j]_{\tilde{Q}}$ to NTT representation for $0 \le j < d$. This procedure requires $d\tilde{\ell}$ unit NTT operations since the NTT operation over $R_{\tilde{Q}}$ requires $\tilde{\ell}$ unit NTT operations and we repeat this operation for each j. In the second step, it requires $2\tilde{\ell}$ unit NTT operations after performing inner products as the inverse NTT operation over $R_{\tilde{Q}}$ takes $\tilde{\ell}$ unit NTT operations and we repeat this operation for each $\tilde{c}_i$ to get its coefficient form. Therefore, the previous key-switching takes total $(d+2)\tilde{\ell}$ unit NTT operations.

We now count the number of unit Hadamard products. In the first step, the base conversion of $b_j = [a]_{D_j}$ to the modulus $\tilde{Q}$ requires $r_j(\tilde{\ell} - r_j) = r(\tilde{\ell} - r) \le r\ell$ unit Hadamard products, and we repeat this procedure for each $0 \le j < d$. Thus, the required number of unit Hadamard products in Step 1 is bounded by $dr\ell = \ell^2$. On the other hand, when $r = 1$, each D_j is a single prime number, and so the first step does not require any Hadamard products. In the second step, it takes $2d\tilde{\ell}$ unit Hadamard products since the Hadamard product over $R_{\tilde{Q}}$ requires $\tilde{\ell}$ unit Hadamard products and each inner product of $\tilde{c}_i = \langle \mathbf{b}, \mathbf{u}_i \rangle \pmod{\tilde{Q}}$ takes d Hadamard products over $R_{\tilde{Q}}$. To sum up, in the case of $r = 1$, the total number of unit Hadamard products required for the previous key-switching operation is $2d\tilde{\ell}$; otherwise, it is $\ell^2 + 2d\tilde{\ell}$.

We note that the gadget decomposition procedure is dominant in the cost of unit NTT operations. The computational cost of unit Hadamard products at the first step is roughly equivalent to the second step except for the case of $r = 1$. As a result, as noted in Sec. 3, the gadget decomposition step turns out to be the most time-consuming part of the previous key-switching operation.

4.3 Our Key-switching Method

We now present our new approach based on the new external product operation. Our key-switching procedure is based on a digit decomposition over $R_{\tilde{Q}}$. For the RNS base $\tilde{\mathcal{B}} = \{q_0, \ldots, q_{\tilde{\ell}-1}\}$ of the modulus $\tilde{Q}$, we choose a partition $\tilde{I}_0, \ldots, \tilde{I}_{\tilde{d}-1}$ of $\{0, 1, \ldots, \tilde{\ell} - 1\}$ and digits $\tilde{D}_j = \prod_{k\in\tilde{I}_j} q_k$. Then, we can define an RNS-friendly gadget decomposition $\tilde{h} : R_{\tilde{Q}} \to R^{\tilde{d}}$ as $\tilde{h}(u) = (v_0, \ldots, v_{\tilde{d}-1})$ for $v_j = [u]_{\tilde{D}_j}$,

and its corresponding gadget vector is $\tilde{\mathbf{g}} = (\tilde{g}_0, \ldots, \tilde{g}_{\tilde{\ell}-1}) \in R_{\tilde{Q}}^{\tilde{d}}$ such that $\tilde{g}_j = 1 \pmod{\tilde{D}_j}$ and $\tilde{g}_j = 0 \pmod{\tilde{D}_{j'}}$ for $j' \neq j$. We generate a key-switching key $\mathbf{U} = [\mathbf{u}_0|\mathbf{u}_1] \in R_{\tilde{Q}}^{d\times 2}$ using the gadget encryption algorithm, and publish the decompositions of $\mathbf{u}_0$ and $\mathbf{u}_1$ with respect to $\tilde{h}$.

Similar to the previous method, our key-switching method consists of three steps. In the first step, the coefficient form of $a \in R_Q$ is given as an input. Then we compute the gadget decomposition $\mathbf{b} = h(a)$ and transform it into the DFT form over the ring R. As discussed in Sec. 3.2, since we instantiate R as $R_{B'}$ in our implementation, it suffices to transform the decomposition into the NTT form over $R_{B'}$. We remark that our method enjoys better performance than the previous approach since the decomposition is computed over $R_{B'}$ rather than $R_{\tilde{Q}}$. Therefore, the complexity of this step mainly depends on the upper bounds of the gadget decompositions h and $\tilde{h}$, which are significantly smaller than the modulus $\tilde{Q}$. In the second step, we compute $\tilde{c}_0 = \langle \mathbf{b}, \mathbf{u}_0 \rangle \pmod{\tilde{Q}}$ and $\tilde{c}_1 = \langle \mathbf{b}, \mathbf{u}_1 \rangle \pmod{\tilde{Q}}$ following the procedure below. First, we perform the inner products $\tilde{c}_{0,j} = \langle \mathbf{b}, \mathbf{v}_{0,j} \rangle$ and $\tilde{c}_{1,j} = \langle \mathbf{b}, \mathbf{v}_{1,j} \rangle$ for $0 \leq j < \tilde{d}$, and convert them back to the coefficient form. Here, we can assume that $\mathbf{V}_0$ and $\mathbf{V}_1$ are given in the NTT form over $R_{B'}$, so these inner products can be computed efficiently over the NTT space of $R_{B'}$. Finally, it holds that

$$\tilde{c}_i = \langle \mathbf{b}, \mathbf{u}_i \rangle = \sum_{0\leq j<\tilde{d}} \langle \mathbf{b}, \mathbf{v}_{i,j} \rangle \cdot \tilde{g}_j = \sum_{0\leq j<\tilde{d}} \tilde{c}_{i,j} \cdot \tilde{g}_j \pmod{\tilde{Q}}$$

from Eq. (1), which implies that $\tilde{c}_i = \tilde{c}_{i,j} \pmod{\tilde{D}_j}$ for all $0 \leq j < \tilde{d}$. Therefore, we have $\tilde{c}_i = \tilde{c}_{i,j} \pmod{q_k}$ for all $0 \leq j < \tilde{d}$ and $k \in \tilde{I}_j$ from the property of digit decomposition, and the RNS representation of $\tilde{c}_i$ modulo $\tilde{Q}$ is obtained as $([\tilde{c}_{i,j}]_{q_k})_{0\leq j<\tilde{d}, k\in\tilde{I}_j} \in \prod_{0\leq k<\tilde{\ell}} R_{q_k}$. The last step of our key-switching is the modulus reduction process, which is identical to that of the previous method.

Our key-switching operation

Input: A polynomial $a \in R_Q$ and two decompositions $\mathbf{V}_i = [\mathbf{v}_{i,0}|\mathbf{v}_{i,1}|\ldots|\mathbf{v}_{i,\tilde{d}-1}]$ of $\mathbf{u}_i$ for $i = 0, 1$, with respect to $\tilde{h}$ for a key-switching key $\mathbf{U} = [\mathbf{u}_0|\mathbf{u}_1] \in R_{\tilde{Q}}^{d\times 2}$.
Step 1 (Decomposition): Compute the gadget decomposition $\mathbf{b} = (b_0, \ldots, b_{d-1}) \leftarrow h(a)$ over R using (2) and (3).
Step 2 (Inner product): Compute $\tilde{c}_{0,j} \leftarrow \langle \mathbf{b}, \mathbf{v}_{0,j} \rangle$ and $\tilde{c}_{1,j} \leftarrow \langle \mathbf{b}, \mathbf{v}_{1,j} \rangle$ over R for $0 \leq j < \tilde{d}$. Compute the elements $\tilde{c}_0$ and $\tilde{c}_1$ of $R_{\tilde{Q}}$ such that $\tilde{c}_0 = \tilde{c}_{0,j} \pmod{D_j}$ and $\tilde{c}_1 = \tilde{c}_{1,j} \pmod{D_j}$ for $0 \leq j < \tilde{d}$.
Step 3 (Mod reduction): Compute $c'_0 = \lfloor \frac{1}{P}\tilde{c}_0 \rceil \pmod{Q}$ and $c'_1 = \lfloor \frac{1}{P}\tilde{c}_1 \rceil \pmod{Q}$.
Output: An RLWE ciphertext $(c'_0, c'_1) \in R_Q^2$.

Complexity analysis. We denote by $\tilde{r}_j = |\tilde{I}_j|$ the length of a digit $\tilde{D}_j$, and assume that all the digit lengths $\tilde{r}_j$ are the same and denoted by $\tilde{r}$, so we get $\tilde{\ell} = \tilde{d}\tilde{r}$. We first count the number of unit NTT operations in our key-switching operation. As noted in Sec. 3.2, we assume that B' is a product of r' distinct word-sized primes. The NTT operation over $R_{B'}$ requires r' unit NTT operations and we repeat this operation for each entry of $\mathbf{b}$ to get its NTT representation, so this step requires dr' unit NTT operations. In the second step, it requires $2\tilde{d}r'$ unit NTT operations after performing inner products since we perform the inverse NTT operation over $R_{B'}$ on $\tilde{c}_{0,j}$ and $\tilde{c}_{1,j}$ to get their coefficient forms. Hence, the total number of unit NTT operations is $(d + 2\tilde{d})r'$.

We now count the number of unit Hadamard products. In the first step, the base conversion of $b_j = [a]_{D_j}$ to the modulus B' requires $r_j r' = rr'$ unit Hadamard products, and we repeat this procedure for each $0 \leq j < d$. Thus, this step requires $drr' = \ell r'$ unit Hadamard products. On the other hand, when $r = 1$, it does not require any Hadamard products. In the second step, the Hadamard product over $R_{B'}$ requires r' unit Hadamard products and each inner product of $\tilde{c}_{i,j} = \langle \mathbf{b}, \mathbf{v}_{i,j} \rangle \pmod{B'}$ takes d Hadamard products over $R_{B'}$. We compute $\tilde{c}_{i,j}$ for $i = 0, 1$ and $0 \leq j < \tilde{d}$, so it takes $2d\tilde{d}r'$ unit Hadamard products in total. At the end of the second step, the base conversion of $[\tilde{c}_{i,j}]_{B'}$ to the modulus q_k requires

r' unit Hadamard products for each $0 \leq i < 2$, $0 \leq j < \tilde{d}$, $0 \leq k \leq \tilde{I}_j$. It follows from the assumption of digit lengths of $\tilde{I}_j$ that this procedure requires $2\tilde{d}\tilde{r}r' = 2\tilde{\ell}r'$ to get the RNS representation of $\tilde{c}_i$ modulo $\tilde{Q}$. To sum up, in the case of $r = 1$, the total number of unit Hadamard products is $(2d\tilde{d} + 2\tilde{\ell})r'$; otherwise, it is $(\ell + 2d\tilde{d} + 2\tilde{\ell})r'$.

We note that the computational cost of unit NTT operations at the first step is roughly equivalent to the second step. On the other hand, the second inner product step is dominant in the cost of unit Hadamard products since only this step is asymptotically quadratic in the level ℓ.

4.4 Complexity Comparison

We now provide an in-depth complexity comparison between previous and our key-switching methods. We analyze the computational complexity of two methods and then compare their space complexity.

Time complexity. Table 1 summarizes the computational complexity of the previous and new key-switching methods in terms of number of unit operations.

Method	**NTT**	**Hadamard Product**	
		$r = 1$	$r > 1$
Previous	$d\tilde{\ell} + 2\tilde{\ell}$	$2d\tilde{\ell}$	$\ell^2 + 2d\tilde{\ell}$
Ours	$dr' + 2\tilde{d}r'$	$2(d\tilde{d} + \tilde{\ell})r'$	$\ell r' + 2(d\tilde{d} + \tilde{\ell})r'$

Table 1: Computational complexity comparison of the previous and our key-switching methods. Here, NTT and Hadamard Product indicate the corresponding unit operations. (d: the dimension of the gadget vector over R_Q, $\tilde{d}$: the dimension of the gadget vector over $R_{\tilde{Q}}$, ℓ: the level of the modulus Q, $\tilde{\ell}$: the number of distinct word-sized primes for the modulus $\tilde{Q} = PQ$, r': the number of distinct word-sized primes for the coefficient bound B')

As mentioned earlier, expensive NTT operations over $R_{\tilde{Q}}$ at the first decomposition step in the previous method are replaced by relatively cheaper NTT operations over $R_{B'}$ in our method (from $d\tilde{\ell}$ to dr' unit NTT operations). In other words, in the gadget decomposition step, a (large) value of $\tilde{\ell}$ in the previous method is reduced to a small value of r' in our method, and therefore the computational complexity of unit NTT operation in this step is reduced by a factor of $\tilde{\ell}/r'$. On the other hand, inverse NTT operations on the inner product results $\tilde{c}_i \in R_{\tilde{Q}}$ in the previous method are roughly equivalent to inverse NTT operations on the inner products $\tilde{c}_{i,j}$ in $R_{B'}$ in our method ($2\tilde{\ell} = 2\tilde{d}\tilde{r}$ vs. $2\tilde{d}r'$ unit NTT operations).

We now consider the cost of unit Hadamard products. The base conversion of an input to the large modulus $\tilde{Q}$ in the previous method requires about ℓ^2 unit Hadamard products, and this term increases significantly for a large level ℓ compared to those of the first and third steps in our method (ℓ^2 vs. $\ell r' + \tilde{\ell}r'$). In the second step, the computational cost of the previous method is roughly equivalent to the new key-switching method ($2d\tilde{\ell}$ vs. $2d\tilde{d}r'$). As a result, the number of unit Hadamard products in our method can be substantially reduced from the previous method as the value of ℓ increases.

To provide further intuition, we rephrase the complexity analysis in an asymptotic manner. By considering $r, \tilde{r}, r' \in O(1)$, we deduce $d, \tilde{d} \in O(\ell)$. Then, both unit NTT operations and Hadamard products in the previous key-switching have an asymptotic complexity of $O(\ell^2)$. Meanwhile, the asymptotic complexity of unit NTT operations in our algorithm is reduced to $O(\ell)$. Therefore, our algorithm reduces the number of NTT operations by a factor of $O(\ell)$ compared to the previous one, while the cost of Hadamard products keeps the same asymptotic computational complexity.

Consequently, our key-switching method provides a considerable speedup over the previous method for an evaluation of large-depth circuits. We note that it is a common practice in HE cryptosystems

to use a sufficiently large value of ℓ, for example in the usage of bootstrapping. In particular, since NTT operations are relatively expensive than Hadamard products and the cost of NTT operations is reduced from quadratic to linear complexity in ℓ, it enables us to achieve a comparably better asymptotic complexity of the key-switching operation.

Space complexity. In terms of space complexity, the key-switching key in the previous key-switching operation is viewed as a $(d \times 2)$ matrix over $R_{\tilde{Q}}$, so the bit-size of a key-switching key is $2d \cdot N \log \tilde{Q} \approx 2d\tilde{d} \cdot N \log \tilde{B}$. Meanwhile, the key-switching key in our new variant consists of two $d \times \tilde{d}$ matrices over R. As noted in Sec. 4.3, it can be given in the NTT form over $R_{B'}$ for efficient computation. Then the bit length of the key-switching key is $2d\tilde{d} \cdot N \log B'$, yielding an expansion factor of $\log B'/\log \tilde{B} \approx \log(2dN \cdot B\tilde{B})/\log \tilde{B} = 1 + \log(2dNB)/\log \tilde{B}$. Alternatively, we can first generate a key-switching key as in the previous method. And then we decompose the key and convert it into an NTT representation before multiplying it by a ciphertext. As a result, we can speed up the key-switching operation while keeping the size of the switching key the same.

5 Implementation and Performance

We present the concrete parameters for the key-switching operation based on the new external product method. Then, we provide experimental results that demonstrate the effectiveness of our new key-switching method. Our source code is based on the CKKS scheme and the implementation is built on top of the Lattigo library [15] that supports the CKKS scheme in a full-RNS manner and with the RNS-based digit decomposition in [22]. The source code is available at `https://github.com/SNUCP/fast-ksw`. All experiments were performed with a single thread on a machine with Intel(R) Xeon(R) Platinum 8268 at 2.90GHz CPU and 192GB RAM running Ubuntu 20.04.3 LTS.

5.1 Parameter Setting

As noted in Sec. 3.2, the coefficient bound B' is a product of r' distinct ν-bit primes that satisfies the inequality $B' > 2dN \cdot B\tilde{B}$ where $B = \frac{1}{2}\max_{0\le j<d}\{D_j\}$ and $\tilde{B} = \frac{1}{2}\max_{0\le j<\tilde{d}}\{\tilde{D}_j\}$. Namely, it suffices to set r' as

$$r' \ge \left\lceil \frac{\log(2dN \cdot B\tilde{B})}{\nu} \right\rceil. \tag{5}$$

In our implementation, the key distribution χ samples each coefficient from $\{0, \pm 1\}$ with probability 0.25 for each of -1 and 1 and probability 0.5 for 0. The error distribution is the discrete Gaussian D_σ with $\sigma = 3.2$. We used two ring dimension parameters $N = 2^{15}$ and $N = 2^{16}$, which are commonly used for an evaluation of a circuit with a sufficiently large depth (e.g., bootstrapping operation). For each of these values of N, we derived an upper bound on the parameter $\tilde{Q}$ to achieve a 128-bit security level. For the modulus $\tilde{Q} = q_0 \cdots q_{\tilde{\ell}-1}$, we set all the primes q_i to be roughly the same size. We set the primes p_j to be $\nu \ge 59.9$ bits as an upper bound for these primes is set to 2^{60} in the Lattigo library. Table 2 summarizes upper bounds on the modulus $\tilde{Q}$ and the level $\tilde{\ell}$, along with approximate sizes for the primes q_i and p_j.

N	$\lceil \log \tilde{Q} \rceil$	$\tilde{\ell}$	$\lceil \log q_i \rceil$	$\lceil \log p_j \rceil$
2^{15}	880	≤ 24	36	60
2^{16}	1761	≤ 48		

Table 2: Parameters used in our experiments

5.2 Experimental Results

We present experimental results of key-switching methods from our experiments. Tables 3 and 4 show the running time (in second) for the previous key-switching operation ("Prev") and the new key-switching operation ("Ours"). To be precise, we measured the time of the key switching operation taken to perform homomorphic multiplication of ciphertexts. We did this for various values of $\tilde{\ell}$ and r while using the parameters in Table 2. We processed the parameter r between 1 and 4 since the default digit length of the gadget decomposition is set as 3 or 4 in the Lattigo library. In each table, the third column gives the level of an input ciphertext by $\ell = \tilde{\ell} - r$. Furthermore, as discussed in Sec. 4.3, our key-switching method relies on the parameters r, $\tilde{r}$, and r'. For each r and $\tilde{r}$, we chose the value of r' by Eq. (5). The last column gives a speedup of the fastest new key-switching operation among experiments with various parameters of $\tilde{r}$ and r' over the previous method.

We now consider the new key-switching method. Given fixed values of N, $\tilde{\ell}$, and r, if taking a large value of $\tilde{r}$, then a large value of r' is chosen by Eq. (5), whereas a small value of $\tilde{d}$ is taken by the fact of $\tilde{d} \approx \tilde{\ell}/\tilde{r}$. So, the first term dr' in the cost of unit NTT operations increases with the parameter $\tilde{r}$. However, because of the decreasing term $r'/\tilde{r}$ with $\tilde{r}$, the second term $2\tilde{d}r'(\approx 2\tilde{\ell}r'/\tilde{r})$ in the cost decreases as $\tilde{r}$ increases. Moreover, the dominant term in the cost of unit Hadamard products is $2d\tilde{d}r' \approx 2d\tilde{\ell}r'/\tilde{r}$, which also decreases with $\tilde{r}$. Therefore, we get an optimal computational complexity when all of these terms are balanced (e.g., $4 \leq \tilde{r} \leq 7$).

The previous key-switching operation is faster for a small value of $\tilde{\ell}$ and a large value of r (due to a small value of $d \approx \ell/r$), which closely aligns with theoretical complexity analysis in Table 1. In Figure 1, the same tendency can be observed in our key-switching operation. Here, the running time is taken as the fastest result among experiments over various $\tilde{r}$ and r' for each $\tilde{\ell}$ and r. Compared to ours, given a fixed value of N, there is much variation in the running time of the previous key-switching operation as $\tilde{\ell}$ increases due to the quadratic and linear dependence of the number of NTTs on $\tilde{\ell}$ in the previous and our methods. Also, the running time of the previous method depends heavily on the value of r than ours.

Given fixed values of N and $\tilde{\ell}$, we get a further speedup of the new key-switching method over the previous method when a small value of r is used. This is because NTT operations are more costly than Hadamard products and the complexity of unit NTT operations for the gadget decomposition in our method is reduced by a factor of $\tilde{\ell}/r'$ compared to the previous method. As the optimal value of r' increases with r, the factor of $\tilde{\ell}/r'$ decreases with r. For example, we get a speedup of 3.3 if $\log N = 16, \tilde{\ell} = 48, r = 1$, and a speedup of 2.2 if $\log N = 16, \tilde{\ell} = 48, r = 4$. On the other hand, the factor of $\tilde{\ell}/r'$ increases with $\tilde{\ell}$. Consequently, our experiments indicate that the speedups for smaller r and larger ℓ are more significant and dramatic.

Micro-benchmarks. To identify the improvements from the impact of the new key-switching method, we present micro-benchmarks for the individual procedures. Figure 2 shows the result with the parameters $N = 2^{16}$ and $\tilde{\ell} = 48$. We do this for various values of r defining the digit length of the gadget decomposition over R_Q. As discussed in Sec. 4.4, the gadget decomposition procedure including the base conversion and NTT operations is the most time-consuming part of the previous key-switching operation. In particular, the cost of NTT operations is dominant in the previous method; the key-switching spends 47–73% of its time performing NTT operations. In contrast, our key-switching procedure spends about 26–27% of its time performing NTT operations and the cost of inner products is dominant in the execution of the procedure. Notably, the NTT computation of our algorithm is around 4–9 times faster than that of the previous algorithm.

Application to homomorphic operations. As mentioned in Sec. 4, key-switching operation is a main bottleneck of homomorphic operations such as multiplication and automorphism. Table 5 shows the performance of multiplication and automorphism with the existing and our key-switching methods. We observe that the previous key-switching method accounts for about 86–93% and 90–95% of the execution times for multiplication and automorphism, respectively. By applying our new key-switching method, we get a speedup of up to 3 times for multiplication and automorphism.

$\log N$	$\tilde{\ell}$	(ℓ, r)		**Prev**	**Ours**						**Speedup**
15	16	$(15,1)$	$\tilde{r}$	-	1	2	3	4	5	6	1.7
			r'	-	2	3	3	4	4	5	
			Time	0.143	0.116	0.101	**0.084**	0.086	0.087	0.092	
		$(14,2)$	$\tilde{r}$	-	2	3	4	5	6	7	1.5
			r'	-	3	4	4	5	6	6	
			Time	0.096	0.074	0.078	**0.062**	0.075	0.076	0.076	
		$(13,3)$	$\tilde{r}$	-	2	3	4	5	6	7	1.2
			r'	-	4	4	5	6	6	7	
			Time	0.077	0.083	0.070	**0.066**	0.077	0.067	0.076	
	20	$(19,1)$	$\tilde{r}$	-	1	2	3	4	5	6	1.9
			r'	-	2	3	3	4	4	5	
			Time	0.216	0.162	0.139	**0.112**	0.118	0.115	0.129	
		$(18,2)$	$\tilde{r}$	-	2	3	4	5	6	7	1.7
			r'	-	3	4	4	5	6	6	
			Time	0.145	0.102	0.103	**0.085**	0.092	0.107	0.093	
		$(17,3)$	$\tilde{r}$	-	2	3	4	5	6	7	1.3
			r'	-	4	4	5	6	6	7	
			Time	0.113	0.109	0.088	**0.087**	0.089	0.088	0.089	
	24	$(23,1)$	$\tilde{r}$	-	1	2	3	4	5	6	2.3
			r'	-	2	3	3	4	4	5	
			Time	0.317	0.214	0.183	**0.140**	0.152	0.142	0.151	
		$(22,2)$	$\tilde{r}$	-	2	3	4	5	6	7	1.9
			r'	-	3	4	4	5	6	6	
			Time	0.204	0.128	0.128	**0.110**	0.118	0.124	0.125	
		$(21,3)$	$\tilde{r}$	-	2	3	4	5	6	7	1.5
			r'	-	4	4	5	6	6	7	
			Time	0.151	0.137	0.106	0.112	0.114	**0.101**	0.115	

Table 3: Experimental results of the previous and our key-switching operations. The evaluation is examined on the full RNS variant of CKKS scheme with the ring dimension $N = 2^{15}$.

6 Conclusion and Future Work

In this paper, we presented a new external product algorithm based on key decomposition technique. Prior to this work, a decomposed ciphertext is converted into the NTT form over a multi-precision modulus and then multiplied with a public evaluation key. We explore another way by refactoring the computation and exploiting the smallness of the gadget decomposition; therefore, we can significantly reduce the number of NTT operation. In our experiment, the key-switching operation based on the new external product method is up to 3 times faster than the previous key-switching operation over typical parameters.

One direction to explore is to take into account hardware implementation, for example, in a GPU environment. According to the recent study by Jung et al. [23], GPU can be used to accelerate the key-switching operation by optimizing the inner product procedure. A significant reduction in NTT operations in our method is a promising possibility that we believe will get a substantial performance improvement when combined with the existing GPU-based optimization techniques.

$\log N$	$\tilde{\ell}$	(ℓ, r)		**Prev**	**Ours**						**Speedup**
16	32	$(31,1)$	$\tilde{r}$	-	1	2	3	4	5	6	2.6
			r'	-	2	3	3	4	4	5	
			Time	1.181	0.744	0.605	0.455	0.476	**0.451**	0.499	
		$(30,2)$	$\tilde{r}$	-	2	3	4	5	6	7	2.4
			r'	-	3	4	4	5	6	6	
			Time	0.854	0.414	0.427	**0.352**	0.399	0.408	0.388	
		$(29,3)$	$\tilde{r}$	-	2	3	4	5	6	7	2.0
			r'	-	4	4	5	6	6	7	
			Time	0.655	0.466	0.341	**0.335**	0.364	0.350	0.372	
		$(28,4)$	$\tilde{r}$	-	2	3	4	5	6	7	1.6
			r'	-	4	5	6	6	7	7	
			Time	0.486	0.403	0.357	0.343	0.319	0.318	**0.301**	
	40	$(39,1)$	$\tilde{r}$	-	1	2	3	4	5	6	3.0
			r'	-	2	3	3	4	4	5	
			Time	1.841	1.098	0.905	0.682	0.711	**0.604**	0.690	
		$(38,2)$	$\tilde{r}$	-	2	3	4	5	6	7	2.7
			r'	-	3	4	4	5	6	6	
			Time	1.378	0.621	0.623	**0.508**	0.531	0.574	0.532	
		$(37,3)$	$\tilde{r}$	-	2	3	4	5	6	7	2.2
			r'	-	4	4	5	6	6	7	
			Time	1.038	0.659	0.521	0.506	0.519	**0.479**	0.507	
		$(36,4)$	$\tilde{r}$	-	2	3	4	5	6	7	2.1
			r'	-	4	5	6	6	7	7	
			Time	0.929	0.581	0.554	0.514	0.456	0.481	**0.438**	
	48	$(47,1)$	$\tilde{r}$	-	1	2	3	4	5	6	3.3
			r'	-	2	3	3	4	4	5	
			Time	2.688	1.532	1.234	0.876	0.938	**0.818**	0.880	
		$(46,2)$	$\tilde{r}$	-	2	3	4	5	6	7	3.0
			r'	-	3	4	4	5	6	6	
			Time	1.980	0.823	0.791	**0.655**	0.702	0.727	0.670	
		$(45,3)$	$\tilde{r}$	-	2	3	4	5	6	7	2.5
			r'	-	4	4	5	6	6	7	
			Time	1.438	0.834	0.629	0.630	0.657	**0.585**	0.617	
		$(44,4)$	$\tilde{r}$	-	2	3	4	5	6	7	2.2
			r'	-	4	5	6	6	7	7	
			Time	1.191	0.738	0.670	0.650	0.589	0.592	**0.550**	

Table 4: Experimental results of the previous and our key-switching operations. The evaluation is examined on the full RNS variant of CKKS scheme with the ring dimension $N = 2^{16}$.

Acknowledgement

This work was supported by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-TB2103-01.

References

1. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full RNS variant of FV like somewhat homomorphic encryption schemes. In: International Conference on Selected Areas in Cryptography. pp. 423–442. Springer

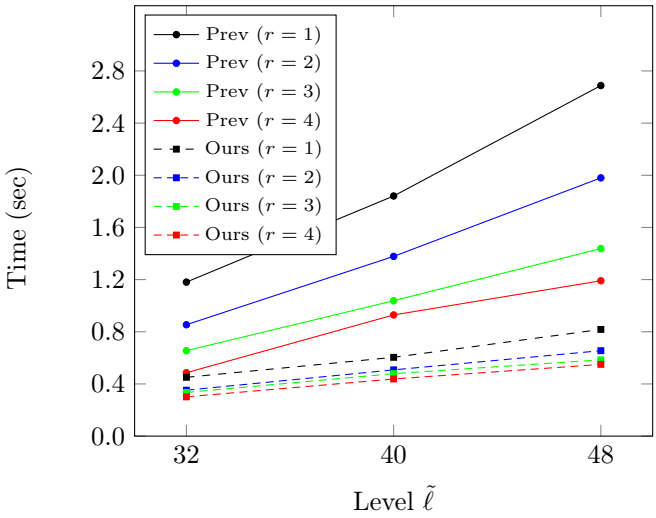

Fig. 1: Running time results of the key-switching methods with $N = 2^{16}$.

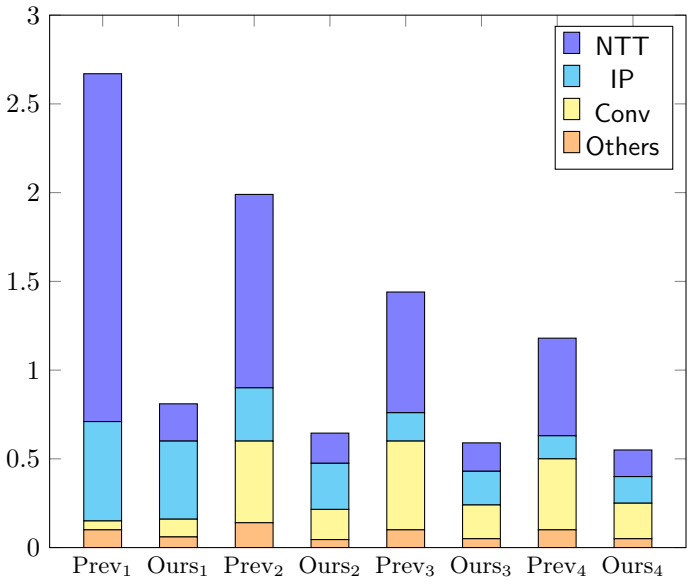

Fig. 2: The detailed proportion of each operation in the previous key-switching (Prev_r) and our key-switching (Ours_r) where $N = 2^{16}$ and $\tilde{\ell} = 48$. NTT stands for both NTT and its inverse, IP stands for inner product, and Conv stands for base conversion.

(2016)
2. Bossuat, J.P., Mouchet, C., Troncoso-Pastoriza, J., Hubaux, J.P.: Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 587–617. Springer (2021)
3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
5. Chen, H., Chillotti, I., Song, Y.: Improved bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 34–54. Springer

$\tilde{\ell}$	r	**Previous**			**Ours**		
		Key-switching	Mult	Auto	Key-switching	Mult	Auto
48	1	2.724 s	2.921 s	2.876 s	0.833 s	1.008 s	0.954 s
	2	1.909 s	2.103 s	2.057 s	0.660 s	0.835 s	0.781 s
	3	1.373 s	1.560 s	1.503 s	0.595 s	0.767 s	0.705 s
	4	1.143 s	1.324 s	1.261 s	0.562 s	0.729 s	0.681 s

Table 5: Experimental results of multiplication and automorphism with the previous and new key-switching operations when $\log N = 16$.

(2019)
6. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 395–412 (2019)
7. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 360–384. Springer (2018)
8. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full RNS variant of approximate homomorphic encryption. In: International Conference on Selected Areas in Cryptography. pp. 347–368. Springer (2018)
9. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)
10. Cheon, J.H., Kim, D., Kim, D.: Efficient homomorphic comparison methods with optimal complexity. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 221–256. Springer (2020)
11. Cheon, J.H., Kim, D., Kim, D., Lee, H.H., Lee, K.: Numerical method for comparison on homomorphically encrypted numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 415–445. Springer (2019)
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
13. DARPA: Data protection in virtual environments (dprive). Online: `https://www.darpa.mil/program/data-protection-in-virtual-environments`
14. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 617–640. Springer (2015)
15. EPFL-LDS: Lattigo v2.3.0. Online: `https://github.com/ldsec/lattigo` (Oct 2021)
16. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing. pp. 169–178. ACM (2009)
18. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
19. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference. pp. 75–92. Springer (2013)
20. Halevi, S., Polyakov, Y., Shoup, V.: An improved RNS variant of the BFV homomorphic encryption scheme. In: Cryptographers' Track at the RSA Conference. pp. 83–105. Springer (2019)
21. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in HElib. In: Annual International Cryptology Conference. pp. 93–120. Springer (2018)
22. Han, K., Ki, D.: Better bootstrapping for approximate homomorphic encryption. In: Cryptographers' Track at the RSA Conference. pp. 364–390. Springer (2020)
23. Jung, W., Kim, S., Ahn, J.H., Cheon, J.H., Lee, Y.: Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with GPUs. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 114–148 (2021)
24. Kim, M., Song, Y., Li, B., Micciancio, D.: Semi-parallel logistic regression for GWAS on encrypted data. BMC Medical Genomics **13**(7), 1–13 (2020)

25. Kim, S., Kim, J., Kim, M.J., Jung, W., Kim, J., Rhu, M., Ahn, J.H.: BTS: An accelerator for bootstrappable fully homomorphic encryption. In: Proceedings of the 49th Annual International Symposium on Computer Architecture. pp. 711–725 (2022)
26. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 618–647. Springer (2021)
27. Lee, Y., Lee, J.W., Kim, Y.S., Kim, Y., No, J.S., Kang, H.: High-precision bootstrapping for approximate homomorphic encryption by error variance minimization. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 551–580. Springer (2022)
28. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 1–23. Springer (2010)
29. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
30. Roy, S.S., Turan, F., Jarvinen, K., Vercauteren, F., Verbauwhede, I.: FPGA-based high-performance parallel architecture for homomorphic computing on encrypted data. In: 2019 IEEE International symposium on high performance computer architecture (HPCA). pp. 387–398. IEEE (2019)
31. Turan, F., Roy, S.S., Verbauwhede, I.: HEAWS: An accelerator for homomorphic encryption on the Amazon AWS FPGA. IEEE Transactions on Computers **69**(8), 1185–1196 (2020)