Optimizing HE operations via Level-aware Key-switching Framework ⋆

Intak Hwang[1], Jinyeong Seo[1], and Yongsoo Song[1]

Seoul National University, Republic of Korea
{intak.hwang, jinyeong.seo, y.song}@snu.ac.kr

Abstract. In lattice-based Homomorphic Encryption (HE) schemes, the key-switching procedure is a core building block of non-linear operations but also a major performance bottleneck. The computational complexity of the operation is primarily determined by the so-called gadget decomposition, which transforms a ciphertext entry into a tuple of small polynomials before being multiplied with the corresponding evaluation key. However, the previous studies such as Halevi et al. (CT-RSA 2019) and Han and Ki (CT-RSA 2020) fix a decomposition function in the setup phase which is applied commonly across all ciphertext levels, resulting in suboptimal performance.
In this paper, we introduce a novel key-switching framework for leveled HE schemes. We aim to allow the use of different decomposition functions during the evaluation phase so that the optimal decomposition method can be utilized at each level to achieve the best performance. A naive solution might generate multiple key-switching keys corresponding to all possible decomposition functions, and sends them to an evaluator. However, our solution can achieve the goal without such communication overhead since it allows an evaluator to dynamically derive other key-switching keys from a single key-switching key depending on the choice of gadget decomposition.
We implement our framework at a proof-of-concept level to provide concrete benchmark results. Our experiments show that we achieve the optimal performance at every level while maintaining the same computational capability and communication costs.

Keywords: Homomorphic Encryption, Gadget Decomposition, Key Switching.

1 Introduction

Homomorphic encryption (HE) is a cryptosystem that enables computation on encrypted data without decrypting it. Since Gentry's the first candidate scheme [15], lattice-based constructions have become a popular method for designing HE schemes. Currently, HE schemes based on the Learning with Errors (LWE) or its ring variant (RLWE) problem [5, 6, 10] are popularly used due to their efficiency and strong security guarantees. These HE schemes introduce a small noise in the encryption algorithm to attain their security which gradually increases after homomorphic operation but also needs to be sufficiently small for correct decryption. To address this issue, the leveled structure has been utilized in several HE schemes (e.g., [6, 10]) where the ciphertext modulus can change dynamically during homomorphic omputation to reduce the noise growth.

Meanwhile, the key-switching operation is a commonly used technique for constructing non-linear HE operations such as multiplication and automorphism. This operation involves multiplying a polynomial (with a large size) and a public (evaluation) key. However, naively multiplying them results in significant noise growth. To address this issue, a special decomposition technique is used, where the polynomial is mapped to a vector of small-sized elements, and the public (evaluation) key is encrypted in a specific vector form. This allows the inner product of the polynomial and the public (evaluation) key to produce the desired result with small noise growth. The mapping and vector are referred to as a gadget decomposition and gadget vector, respectively, and we also refer to the pair of them as a gadget toolkit in this paper.

⋆ This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [NO.2022-0-01047, Development of statistical analysis algorithm and module using homomorphic encryption based on real number operation].

The Residue Number System (RNS) is widely used in HE implementations to support high-precision arithmetic efficiently. For example, there have been several studies [3, 9, 16, 17] on RNS-friendly design and implementation of the B/FV [5, 14] and CKKS [10] schemes, and these results form the basis of state-of-the-art HE libraries (e.g., [1, 7, 26]). Yet, the key-switching operation remains the most time-consuming part in HE operations.

Our Contributions. In this paper, we introduce a new key-switching method for leveled HE schemes, which we call the *level-aware* key-switching framework. The main advantage of our algorithm is its ability to select the appropriate gadget toolkit at each level, which was not possible in previously.

The existing method uses a single gadget toolkit for all levels despite the fact that the best-performing gadget toolkit is different for each level. This is because manipulating the underlying gadget toolkit is cumbersome, as the gadget vector part is present in an encrypted form entangled with noises. To handle this issue, we observe the algebraic properties of gadget vectors, and devise a new way to manipulate the underlying gadget vectors of public (evaluation) keys in an encrypted state. Consequently, leveraging the newly observed properties, we build a new key-switching framework that can control the underlying gadget toolkit based on the input ciphertext level.

Another way to address advantages of our framework is reduction in communication cost. In the previous framework, a key generator should send additional key-switching keys to an evaluator, incurring additional communication costs, to achieve equivalent performance with ours. However, our framework effectively eliminates this overhead, leading to better key-switching performance under the same communication cost during the key generation phase.

We note that our method is a pure algorithmic optimization, so it does not affect the security of the underlying schemes. Moreover, our algorithm is not limited to the CKKS scheme, and can be applied to other lattice-based leveled HE schemes, such as the BGV [6] and the leveled BFV [18] schemes. Furthermore, our algorithm is compatible with RNS-based implementations, so one can directly apply our method to the existing state-of-the-art HE libraries such as OpenFHE [1], SEAL [7], and Lattigo [26].

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 $a = \sum_{0\le i<N} a_i X^i$ of R (or R_q) with the vector of its coefficients $(a_0, \ldots, a_{N-1})$ in $\mathbb{Z}^N$ (or $\mathbb{Z}_q^N$, resp). 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 each coefficient of $a \in R$ 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 distribution $\mathcal{D}$, we use $x \leftarrow \mathcal{D}$ to denote the sampling x according to $\mathcal{D}$. For a finite set S, we denote the uniform distribution over S as $\mathcal{U}(S)$. 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 N be a power of two, Q a positive integer, and χ and ψ be distributions over R. The ring learning with errors (RLWE) assumption with respect to the parameter (N, Q, χ, ψ) is that given polynomially many samples of either (b, a) or $(-as + e, a)$, where $a, b \leftarrow R_Q$, $s \leftarrow \chi$, $e \leftarrow \psi$, it is computationally hard to distinguish which is the case. The security of lattice-based homomorphic encryption (HE) schemes such as BGV [6], B/FV [5, 14], and CKKS [10] relies on the hardness of the RLWE assumption.

Throughout this paper, we refer a pair $(c_0, c_1) \in R_Q^2$ as an RLWE ciphertext encrypted under s, if $c_0 = -c_1 s + \mu + e \pmod{Q}$ for some $\mu, s \in R$ and small e. Then, it admits $c_0 + c_1 s \approx \mu \pmod{Q}$.

2.2 RNS and NTT

When implementing HE schemes, we need big-integer arithmetic as HE ciphertexts usually have large modulus. To handle this issue, the Residue Number System (RNS) is popularly used, where a large integer is represented as a tuple of small integers. There have been several literature [9, 16, 17, 18] regarding efficient implementation of HE schemes based on the RNS representation. The RNS representation is defined as follows. Let $Q = q_0 \cdots q_{\ell-1}$ be a big integer where q_i'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. We refer to the image $([a]_{q_0}, \ldots, [a]_{q_{\ell-1}})$ as the RNS representation of a. With q_i set as word-size primes, arithmetic operations over R_Q can be efficiently instantiated using R_{q_i}, avoiding the need for inefficient high-precision arithmetic.

However, polynomial multiplications in R_{q_i} still need to be optimized since its naive implementation takes $O(N^2)$ complexity. Hence, to accelerate polynomial multiplications in R_{q_i}, the discrete Fourier transformation over $\mathbb{Z}_{q_i}$, also known as Number Theoretic Transform (NTT), is frequently utilized. If $q_i = 1 \pmod{2N}$, there exists a $(2N)$-th root of unity $\zeta_i \in \mathbb{Z}_{q_i}$. Then, we obtain a ring isomorphism from R_{q_i} to $\mathbb{Z}_{q_i}^N$ defined by $a \mapsto (a(\zeta_i), a(\zeta_i^3), \ldots, a(\zeta_i^{2N-1}))$. This isomorphism is called the NTT conversion modulo q_i. We note that the NTT conversion (or its inverse) can be computed in $O(N \log N)$ arithmetic operations over $\mathbb{Z}_{q_i}$. Using this conversion, one can instantiate polynomial multiplication with $O(N \log N)$ complexity as follows. First, transform operand polynomials to $\mathbb{Z}_{q_i}^N$ using the NTT, then apply point-wise multiplication to them in $\mathbb{Z}_{q_i}^N$, and finally, transform the result back to the polynomial ring R_{q_i} using the inverse NTT.

2.3 Gadget Decomposition and Key-switching

We first review the definition of the *gadget toolkit*, which is commonly used as building blocks for lattice-based HE schemes [6, 14, 13, 10].

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 $\mathbf{b} = (b_0, b_1, \ldots, b_{d-1}) \leftarrow h(a)$:*

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

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

Given a gadget toolkit over $(h, \mathbf{g})$ over R_Q, A *gadget encryption* of a plaintext $\mu \in R$ under a secret key s and a special modulus P is defined as $(\mathbf{u}_0, \mathbf{u}_1) \in R_{PQ}^{d \times 2}$ where $\mathbf{u}_0 = -s \cdot \mathbf{u}_1 + P\mu \cdot \mathbf{g} + \mathbf{e} \pmod{PQ}$ for some small $\mathbf{e} \in R^d$. Then, for $a \in R_Q$, we can compute a ciphertext

$$(c_0, c_1) = \left\lfloor \frac{1}{P}(\langle h(a), \mathbf{u}_0 \rangle, \langle h(a), \mathbf{u}_1 \rangle) \right\rceil$$

whose decryption is approximately equal to $\mu \cdot a$. This can be checked as follows.

$$\begin{aligned} c_0 + c_1 \cdot s &= \frac{1}{P} \langle h(a), \mathbf{u}_0 + s \cdot \mathbf{u}_1 \rangle + e_0 + e_1 s \pmod{Q} \\ &= \mu \cdot a + \frac{1}{P} \langle h(a), \mathbf{e} \rangle + e_0 + e_1 s \pmod{Q} \end{aligned} \tag{1}$$

where $e_0 = \left\lfloor \frac{1}{P} \langle h(a), \mathbf{u}_0 \rangle \right\rceil - \frac{1}{P} \langle h(a), \mathbf{u}_0 \rangle$ and $e_1 = \left\lfloor \frac{1}{P} \langle h(a), \mathbf{u}_1 \rangle \right\rceil - \frac{1}{P} \langle h(a), \mathbf{u}_1 \rangle$ are errors from rounding. Note that we have
$\left\| \frac{1}{P} \langle h(a), \mathbf{e} \rangle \right\|_\infty \le \frac{B}{P} N \cdot \|\mathbf{e}\|_\infty$, thus if P is large enough so that B/P is small, we obtain $c_0 + c_1 s \approx \mu \cdot a$ $\pmod{Q}$.

The above procedure is referred as a *key-switching* operation, and is frequently utilized to construct non-linear homomorphic operations in lattice-based HE schemes. We also call a gadget encryption as a key-switching key.

2.4 Leveled Homomorphic Encryption

We provide a brief description of CKKS [10] as an example of leveled HE scheme.

• $\underline{\texttt{CKKS.Setup}(1^{\lambda})}$: Given a security parameter λ, it chooses a RLWE parameter (N, PQ, χ, ψ). Additionally, it chooses a maximum level $\ell_{\max}$, a chain of ciphertext moduli $Q_1 | Q_2 \dots | Q_{\ell_{\max}} = Q$, a gadget toolkit $(h, \mathbf{g})$ for R_Q, and a special modulus P, scaling factor $\mathit{\Delta}$, then it outputs the public parameter $\mathsf{pp} = (N, Q_1, \dots, Q_{\ell_{\max}}, P, h, \mathbf{g}, \chi, \psi, \mathit{\Delta})$.

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

- Sample $s \leftarrow \chi$ and return the secret key $\mathsf{sk} = s$.
- Sample $p_1 \leftarrow R_{PQ}$ and $e_p \leftarrow D_{\sigma}$, and let $\mathbf{p} = (p_0, p_1) \in R_{PQ}^2$ where $p_0 = -p_1 s + e_p \pmod{\tilde{Q}}$. Return the public key $\mathsf{pk} = \mathbf{p}$.
- Sample $\mathbf{r}_1 \leftarrow R_{PQ}^d$ and $\mathbf{e}_r \leftarrow D_{\sigma}^d$. Generate the relinearization key as $\mathsf{rlk} \leftarrow (\mathbf{r}_0, \mathbf{r}_1) \in R_{PQ}^{d \times 2}$ where $\mathbf{r}_0 = -s \cdot \mathbf{r}_1 + Ps^2 \cdot \mathbf{g} + \mathbf{e}_r \pmod{PQ}$.

• $\underline{\texttt{CKKS.Enc}(\mathsf{pk}; m)}$: Given a public key pk and a message $m \in \mathbb{R}[X]/(X^N + 1)$, it outputs a ciphertext $\mathsf{ct} = \left\lfloor \frac{1}{P} \cdot \left(z \cdot \mathsf{pk} + (e_0, e_1)\right) \right\rceil + \left(\lfloor \mathit{\Delta} m \rceil, 0\right) \pmod{Q_\ell}$ of level ℓ where $z \leftarrow \chi$ and $e_0, e_1 \leftarrow \psi$.

• $\underline{\texttt{CKKS.Dec}(s; \mathsf{ct})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_{Q_\ell}^2$ and a secret key $s \in R$, it outputs a message $m = \frac{1}{\mathit{\Delta}}(c_0 + s \cdot c_1 \pmod{Q_\ell}) \in \mathbb{R}[X]/(X^N + 1)$.

• $\underline{\texttt{CKKS.Rescale}(\mathsf{ct})}$: Given a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_{Q_\ell}^2$ of level ℓ, it outputs a ciphertext $\mathsf{ct}' = \left(\left\lfloor \frac{Q_{\ell-1}}{Q_\ell} c_0 \right\rceil, \left\lfloor \frac{Q_{\ell-1}}{Q_\ell} c_1 \right\rceil\right) \in R_{Q_{\ell-1}}^2$ of level $\ell - 1$.

• $\underline{\texttt{CKKS.Add}(\mathsf{ct}_1, \mathsf{ct}_2)}$: Given two ciphertexts $\mathsf{ct}_1 \in R_{Q_{\ell_1}}^2, \mathsf{ct}_2 \in R_{Q_{\ell_2}}^2$ of level ℓ_1 and ℓ_2 respectively, it outputs a ciphertext $\mathsf{ct}_{add} = \mathsf{ct}_1 + \mathsf{ct}_2 \pmod{Q_\ell}$ of level ℓ where $\ell = \min\{\ell_1, \ell_2\}$.

• $\underline{\texttt{CKKS.Mult}(\mathsf{rlk}; \mathsf{ct}_1, \mathsf{ct}_2)}$: Given two ciphertexts $\mathsf{ct}_1 = (c_0, c_1) \in R_{Q_{\ell_1}}^2$, $\mathsf{ct}_2 = (c_0', c_1') \in R_{Q_{\ell_2}}^2$ of level ℓ_1 and ℓ_2 respectively and a relinearization key $\mathsf{rlk} \in R_{PQ}^{d \times 2}$, let $d_0 = c_0 c_0' \pmod{Q_\ell}$, $d_1 = (c_0 c_1' + c_0' c_1) \pmod{Q_\ell}$, and $d_2 = c_1 c_1' \pmod{Q_\ell}$ where $\ell = \min\{\ell_1, \ell_2\}$. Then, it outputs a ciphertext $\mathsf{ct}_{mul} = (d_0, d_1) + \left\lfloor \frac{1}{P}(\langle h(d_2), \mathsf{rlk}_0 \rangle, \langle h(d_2), \mathsf{rlk}_1 \rangle) \right\rceil \pmod{Q_\ell}$ of level ℓ.

We note that the generation of relinearization key and the homomorphic multiplication algorithm in the above implicitly involves gadet enrpytion and key-switching operation respectively. After the homomorphic multiplication, the scaling factor of $\mathsf{ct}_{\mathrm{mul}}$ becomes $\mathit{\Delta}^2$, which is undesirable since repeated multiplications would yield exponential blow-up of scaling factor. To overcome this problem, the rescaling operation is performed after each multiplication. If we set $\mathit{\Delta} \approx Q_\ell / Q_{\ell-1}$ in the setup phase, then the rescaling operation reduces the scaling factor of plaintext by the factor of $\mathit{\Delta}$ at the expense of one level. For more details on the CKKS scheme such as homomorphic automorphism or the correctness proof, we refer the reader to [10].

3 Revisiting RNS-based Decomposition

There have been several work [9, 16, 17, 18] that studied an efficient implementation of gadget toolkits using RNS representation. These gadget toolkits are often referred to as RNS-based gadget toolkits due to their compatibility with RNS representation. Since they provide efficient performance in real-world implementations, they are popularly used in most HE libraries [7, 26, 1].

In this section, we revisit these RNS-based gadget toolkits and present our new observations regarding them. In short, our observation is related to the transformation between RNS-based gadget toolkits. To state it more clearly, we start by reviewing RNS-based gadget toolkits. Then, we demonstrate how an RNS-based gadget toolkit can be transformed into another gadget toolkit through simple additions.

3.1 Previous Method

To begin with, we define some notations regarding the RNS-based gadget toolkit. Let $q_0, q_1, \ldots, q_{\ell-1}$ be a pairwise coprime integers and $Q = \prod_{i=0}^{\ell-1} q_i$. Let $0 = t_0 < t_1 < \cdots < t_{d-1} < t_d = \ell$ be a sequence of integers. For $0 \le j < d$, we define the j-th digit as $D_j := \prod_{t_j \le i < t_{j+1}} q_i$ and write $\hat{D}_j = Q/D_j$. Then, the RNS-based decomposition with respect to digits D_j is defined as

$$h(a) = \left([\hat{D}_0^{-1} \cdot a]_{D_0}, \ldots, [\hat{D}_{d-1}^{-1} \cdot a]_{D_{d-1}} \right).$$

We can show that h is a gadget decomposition corresponding to the gadget vector $\mathbf{g} = (\hat{D}_0, \ldots, \hat{D}_{d-1})$ from the Chinese Remainder Theorem.

We note that $h(a)$ can be computed in the RNS representation using the *base conversion* algorithm from the following formula:

$$[\hat{D}_j^{-1} \cdot a]_{D_j} = \sum_{t_j \le i < t_{j+1}} [(Q/q_i)^{-1} \cdot a]_{q_i} \cdot (D_j/q_i) - r_j \cdot D_j$$

where $r_j = \left\lfloor \sum_{t_j \le i < t_{j+1}} q_i^{-1} \cdot [(Q/q_i)^{-1} \cdot a]_{q_i} \right\rceil \in R$. We refer the reader to [17, 18] for more details.

3.2 Coalescing Gadget Vector

In this section, we present a variant of the RNS-based gadget decomposition method and explain its properties. At a high level, we demonstrate how to trjnsform an RNS-based decomposition into another with a different RNS basis and describe the relation between their gadget vectors.

To be precise, we show that performing simple additions between elements of an RNS-based gadget vector leads to an efficient transformation that coalesce digits. Suppose that h is an RNS-based decomposition with respect to a basis $D_0, \ldots, D_{d-1}$. For an integer $k > 0$, let $\bar{D}_j = \prod_{jk \le i < (j+1)k} D_i$ for $0 \le j < \bar{d} = \lceil d/k \rceil$. Then, the function $\bar{h} : R_Q \to R^{\bar{k}}$ defined by

$$\bar{h}(a) = \left(\left[\Big(\sum_{j=0}^{k-1} \hat{D}_j\Big)^{-1} \cdot a \right]_{\bar{D}_0}, \ldots, \left[\Big(\sum_{j=(\bar{d}-1)k}^{d-1} \hat{D}_j\Big)^{-1} \cdot a \right]_{\bar{D}_{\bar{k}-1}} \right)$$

becomes a gadget decomposition corresponding to the vector

$$\bar{\mathbf{g}} = \left(\sum_{j=0}^{k-1} \hat{D}_j, \ldots, \sum_{j=(\bar{d}-1)k}^{d-1} \hat{D}_j \right).$$

To verify our claim, we first remark that $\sum_{i=jk}^{(j+1)k-1} \hat{D}_i$ is invertible in $\mathbb{Z}_{\bar{D}_j}$ so the decomposition $\bar{h}$ is well-defined. In addition, we have $\langle \bar{h}(a), \bar{\mathbf{g}} \rangle = [a]_{\bar{D}_j} \pmod{\bar{D}_j}$ for $0 \le j < \bar{d}$, which implies $\langle \bar{h}(a), \bar{\mathbf{g}} \rangle = a \pmod{Q_\ell}$ from the Chinese Remainder Theorem. The above observation allows us to manipulate the underlying structure of key-switching keys since key-switching keys are essentially encryptions of gadget vectors multiplied with some messages. We elaborate on this discussion in the next section.

4 Level-aware Key-switching Framework

In this section, we present a new framework for key-switching operations in leveled HE schemes, which we call *level-aware* key-switching framework. The main objective of this section is to investigate how we can modify underlying structure of key-switching key, such as the special modulus or gadget vector.

In a nutshell, we aim to utilize the observation in Sec. 3.2 to enhance key-switching performance. As previously discussed, one can modify the digits in an RNS-based gadget vector by simply adding each component. We apply a similar technique to key-switching keys to alter the digits in their underlying gadget vectors, enabling us to derive a wide range of key-switching keys with different compositions of digits. Consequently, this allows us to achieve more significant improvements in the performance of key-switching operations in a leveled setting by selecting the best-performing key-switching key at each level.

In the rest of this section, we first revisit the previous key-switching framework. Then, we discuss how we can adapt the newly observed property to construct the new key-switching framework. We also assume that $q_0, \ldots, q_{L-1}$ are distinct, similar-sized primes, comprising a chain of moduli $Q_\ell = \prod_{i=0}^{\ell-1} q_i$ for $0 \leq \ell < L$. Additionally, we assume that an RLWE parameter is given as (N, Q_L, χ, ψ) with respect to a security parameter λ.

4.1 Previous Key-switching Procedure

Below, we review the basics of the key-switching procedure utilized in previous literature.

- $\underline{\texttt{Setup}(r)}$: Given a digit length r, set the maximum ciphertext level $\ell_{\max} = L - r$ and the maximum gadget dimension $d_{\max} = \lceil \ell_{\max}/r \rceil$. Also, set the j-th digit as $D_j = \prod_{i=jr}^{(j+1)r-1} q_i$ for $0 \leq j < d_{\max}$, and set the special modulus as $P = Q_L/Q_{L-r}$.

- $\underline{\texttt{KSKGen}(\mu; s)}$: Given a message $\mu \in R$ and a secret key $s \in R$, output a key-switching key $(\mathbf{u}_0, \mathbf{u}_1) \in R_{Q_L}^{d_{\max} \times 2}$ where $\mathbf{u}_1 \leftarrow \mathcal{U}(R_{Q_L}^{d_{\max}})$, $\mathbf{e} \leftarrow \psi^{d_{\max}}$, and $\mathbf{u}_0 = -s \cdot \mathbf{u}_1 + \mu \cdot \big(Q_L/D_0, \ldots, Q_L/D_{d_{\max}-1}\big) + \mathbf{e} \pmod{Q_L}$.

- $\underline{\texttt{KS}(a; \mathsf{ksk})}$: Given a polynomial $a \in R_{Q_\ell}$ of level ℓ and a key-switching key $\mathsf{ksk} = (\mathbf{u}_0, \mathbf{u}_1)$, output a ciphertext (c_0, c_1) of level ℓ computed as follows:

1. Parse $\mathbf{u}_i$ as $(u_{i,0}, \ldots, u_{i,d_{\max}-1})$ for $i = 0, 1$, and set $d = \lceil \ell/r \rceil$.
2. $\mathbf{b} \leftarrow \big([(Q_{\ell_{\max}}/D_0)^{-1} a]_{D_0}, \ldots, [(Q_{\ell_{\max}}/D_{d-1})^{-1} a]_{D_{d-1}}\big)$.
3. $\mathbf{v}_i \leftarrow ([u_{i,0}]_{PQ_\ell}, \ldots, [u_{i,d-1}]_{PQ_\ell})$ for $i = 0, 1$.
4. $c_i \leftarrow \left\lfloor \frac{1}{P} \langle \mathbf{b}, \mathbf{v}_i \rangle \right\rceil \pmod{Q_\ell}$ for $i = 0, 1$.

At the setup phase, a key generator designates a digit length r that will be used throughout the whole process. Then, the maximum ciphertext level $\ell_{\max}$, a special modulus P, and digits D_j are determined accordingly. For key-switching key generation, it essentially computes a gadget encryption of μ with respect to the special modulus $P = \frac{Q_L}{Q_{\ell_{\max}}}$ and a gadget vector $\left(\frac{Q_{\ell_{\max}}}{D_0}, \ldots, \frac{Q_{\ell_{\max}}}{D_{d_{\max}-1}}\right)$. Note that $P \cdot \frac{Q_{\ell_{\max}}}{D_j} = \frac{Q_L}{D_j}$, so we encrypt $\mu \cdot \frac{Q_L}{D_j}$'s at the keygen phase. Finally, for key-switching operation, it first computes the gadget decomposition with respect to the level of an input polynomial, takes modulo operations on the input key-switching keys, and performs an inner product of them.

To present how the value of r affects the performance of key-switching, we provide a computational complexity analysis of key-switching. In the analysis, we primarily count the number of two operations: NTT (Number Theoretic Transform) or inverse NTT over R_{q_i} and the Hadamard product (pointwise product) over $\mathbb{Z}_{q_i}^N$ asthey dominate the overall elapsed time. Table 1 provides a summary of the analysis obtained in [20]. For details, we refer to the reference.

NTT	**Hadamard Product**
$O(d\ell + r)$	$O(\ell^2 + d\ell)$

Table 1: Computational complexity of each operation in key-switching. ℓ and r denotes input level and digit length respectively, and $d = \lceil \ell/r \rceil$

We note that the previous key-switching algorithm $\texttt{KS}$ adaptively adjusts the dimension of gadget decomposition and the modulus of key-switching keys depending on the level of input ciphertexts. This improves key-switching performance by reducing the value of d since smaller d value gives better performance as described in Table. 1. However, it does not lead to significant speed-up in key-switching since impact of changes in d is small as r remains fixed.

To sum up, in the previous key-switching, the entire process is determined by the digit length chosen during the setup phase by a key generator. As a result, an evaluator's role in key-switching operations is somewhat passive in that it simply applies modulo operations on key-switching keys depending on the level of the input.

4.2 Coalescing Key-switching Key

Our new framework stems from the observation in Sec. 3.2, which shows that a gadget vector with larger digits can be obtained by summing components of a gadget vector with smaller digits. A natural way to apply this method to key-switching keys is to add each component of key-switching keys. However, this may not result in well-formed key-switching keys as we need to adjust the special modulus with respect to the changed digits for correct key-switching operations. To be precise, consider a key-switching key of the previous framework generated with $(\mathbf{u}_0, \mathbf{u}_1) \leftarrow \texttt{KSKGen}(\mu; s)$. It can be regarded as RLWE encryptions of

$$P\mu \cdot \left(\frac{Q_{\ell_{\max}}}{D_0}, \ldots, \frac{Q_{\ell_{\max}}}{D_{d_{\max}-1}} \right).$$

where $P = Q_L / Q_{\ell_{\max}}$. Following the observation in Sec. 3.2, if we add k successive key-switching components, it results in RLWE encryptions of

$$P\mu \cdot \left(\sum_{j=0}^{k-1} \frac{Q_{\ell_{\max}}}{D_j}, \ldots, \sum_{j=(\lceil d_{\max}/k \rceil - 1)k}^{d_{\max}-1} \frac{Q_{\ell_{\max}}}{D_j} \right). \tag{2}$$

Then, we obtain a key-switching key encrypted from the special modulus P and a gadget vector with a kr-digit length. However, as the digit length grows from r to kr, an upper bound for gadget decomposition grows as well. Thus, the special modulus P may not be sufficient for compensating key-switching noises.

To address this issue, we further observe the structure of key-switching keys and discover that unused modulus of higher levels can be transferred to the special modulus at lower levels. To be precise, we observe that each component of Eq. (2) can be represented as follows:

$$P\mu \cdot \sum_j \frac{Q_{\ell_{\max}}}{D_j} = \frac{P Q_{\ell_{\max}}}{Q_\ell} \mu \cdot \sum_j \frac{Q_\ell}{D_j}$$

Hence, if we regard the maximum ciphertext level as ℓ, then Eq. (2) can be considered as a gadget encryption with the special modulus $\frac{PQ_{\ell_{\max}}}{Q_\ell}$, where the unused modulus $\frac{Q_{\ell_{\max}}}{Q_\ell} = q_{\ell_{\max}-1} \cdots q_\ell$ is transferred to the special modulus.

This allows us to derive valid key-switching keys with different compositions of digits from the original one as the level goes down. By leveraging this property, we come up with a new framework for key-switching operations that dynamically selects the best-performing key-switching keys at each level, which we call the *level-aware* framework.

4.3 Level-aware Key-switching Framework

Below, we present our new *level-aware* key-switching framework.

• $\underline{\texttt{Setup}(\cdot)}$: For each digit length r, set the maximum ciphertext level $\ell_{\max}^{(r)} = L - r$ and the maximum gadget dimension $d_{\max}^{(r)} = \left\lceil \ell_{\max}^{(r)} / r \right\rceil$. Also, set the j-th digit as $D_j^{(r)} = \prod_{i=jr}^{(j+1)r-1} q_i$ for $0 \le j < d_{\max}^{(r)}$, and set the special modulus as $P^{(r)} = Q_L / Q_{L-r}$.

- $\underline{\texttt{KSKGen}(\mu; s)}$: Given a message $\mu \in R$ and a secret key $s \in R$, output a key-switching key $(\mathbf{u}_0, \mathbf{u}_1) \in R_{Q_L}^{\ell_{\max} \times 2}$ where $\mathbf{u}_1 \leftarrow \mathcal{U}(R_{Q_L}^{\ell_{\max}})$, $\mathbf{e} = (e_0, \dots, e_{\ell_{\max}-1}) \leftarrow \varphi^{\ell_{\max}}$, and $\mathbf{u}_0 = -s \cdot \mathbf{u}_1 + \mu \cdot \big(Q_L/q_0, \dots, Q_L/q_{\ell_{\max}-1}\big) + \mathbf{e} \pmod{Q_L}$.

- $\underline{\texttt{Expand}(\mathsf{ksk}; I)}$: Given a key-switching key $\mathsf{ksk} = (\mathbf{u}_0, \mathbf{u}_1)$ and an index map I, output expanded key-switching keys $\{\mathsf{ksk}^{(r)} = (\mathbf{u}_0^{(r)}, \mathbf{u}_1^{(r)})\}_{r \in \mathrm{ran}(I)}$ which are computed as follows:

  1. Parse $\mathbf{u}_i$ as $(u_{i,0}, \dots, u_{i,\ell_{\max}-1})$ for $i = 0, 1$.
  2. Set $d = d_{\max}^{(r)}$, and $D_j = \prod_{k=jr}^{(j+1)r-1} q_k$ for $0 \leq j < d$.
  3. $\mathbf{u}_i^{(r)} \leftarrow \left(\sum_{k=0}^{r-1} u_{i,k}, \dots, \sum_{k=(d-1)r}^{\ell_{\max}-1} u_{i,k}\right)$ for $i = 0, 1$.

- $\underline{\texttt{KS}(a; \{\mathsf{ksk}^{(r)}\}_{r \in \mathrm{ran}(I)})}$: Given a polynomial $a \in R_{Q_\ell}$ of level ℓ and expanded key-switching keys $\{\mathsf{ksk}^{(r)} = (\mathbf{u}_0^{(r)}, \mathbf{u}_1^{(r)})\}_{r \in \mathrm{ran}(I)}$, output a ciphertext (c_0, c_1) of level ℓ computed as follows:

  1. Set the digit length $r = I(\ell)$ for the level $\ell \leq \ell_{\max}^{(r)}$
  2. Parse $\mathbf{u}_i^{(r)}$ as $\left(u_{i,0}^{(r)}, \dots, u_{i,d_{\max}^{(r)}-1}^{(r)}\right)$ for $i = 0, 1$, and $d = \lceil \ell/r \rceil$.
  3. $\mathbf{b} \leftarrow \left(\left[\left(\sum_{k=0}^{r-1} \frac{Q_{L-r}}{q_k}\right)^{-1} a\right]_{D_0^{(r)}}, \dots, \left[\left(\sum_{k=(d-1)r}^{\ell-1} \frac{Q_{L-r}}{q_k}\right)^{-1} a\right]_{D_{d-1}^{(r)}}\right)$
  4. $\mathbf{v}_i \leftarrow ([u_{i,0}^{(r)}]_{P^{(r)}Q_\ell}, \dots, [u_{i,d-1}^{(r)}]_{P^{(r)}Q_\ell})$ for $i = 0, 1$.
  5. $c_i \leftarrow \left\lfloor \frac{1}{P^{(r)}} \langle \mathbf{b}, \mathbf{v}_i \rangle \right\rceil \pmod{Q_\ell}$ for $i = 0, 1$.

The basic pipeline of our framework is as follows. At the setup phase, parameters such as the maximum ciphertext level, digits, and the special modulus are computed for each digit length r since our framework utilizes various digit lengths depending on the level. Then, a key generator generates initial key-switching keys with a single digit so that an evaluator can freely obtain the best-performing key as needed.

Upon receiving initial key-switching keys from a key generator, an evaluator preprocesses these keys before the evaluation phase. First, the evaluator investigates the best-performing digit lengths for each level and constructs an index map $I : [L] \to [L]$, where $I(\ell)$ represents the optimal digit length for the level ℓ. Subsequently, the evaluator expands the key-switching keys with respect to I using the key expansion algorithm `Expand`, ensuring that the best-performing key-switching keys are prepared ahead of the evaluation phase. Once the preprocessing is complete, the evaluator enters the evaluation phase and executes the `KS` algorithm, which performs the key-switching operation using the best-performing key-switching keys based on the input level.

Compared to the previous key-switching method, the computational complexity analysis remains unchanged as our method does not alter the structure of the key-switching algorithm itself. However, the main difference lies in the capability to alter the digit length r of the underlying gadget vector for key-switching keys, which directly impacts the performance of key-switching operations, as shown in Table 1.

Hence, the complexity analysis helps us to construct an index map I for optimal performance. For example, when ℓ is sufficiently larger than the possible r values, a larger r results in better performance since it effectively reduces the size of d. As a result, we can achieve a significant speed-up in key-switching operations at high levels by mapping $I(\ell)$ to $L - \ell$. However, for small ℓ values, asymptotic analysis is not very helpful, so we empirically determine optimal $I(\ell)$ values based on benchmark results.

On the other hand, our framework requires an evaluator to execute a preprocessing procedure for key-switching keys, incurring additional computational costs. However, in most cases, this preprocessing is executed at most once since the derived key-switching keys can be reused throughout the entire evaluation phase. Thus, the performance enhancements in the evaluation phase compensate for this initial overhead. We elaborate on these discussions in the next section, along with concrete experimental results.

We end this section by providing a correctness proof and noise analysis on our new framework, especially for the key expansion algorithm `Expand` and the key-switching algorithm `KS`.

Correctness. We first show that for level $\ell < L$ and any digit length r satisfying $\ell + r \leq L$, a key-switching key $(\mathbf{u}_0^{(r)}, \mathbf{u}_1^{(r)})$ expanded from $(\mathbf{u}_0, \mathbf{u}_1)$ with the algorithm `Expand` provides correct key-switching functionality when performed with the `KS` algorithm. Suppose $(\mathbf{u}_0, \mathbf{u}_1) \leftarrow \texttt{KSKGen}(\mu; s)$ for some $\mu, s \in R$, then $(\mathbf{u}_0^{(r)}, \mathbf{u}_1^{(r)}) \leftarrow \texttt{Expand}((\mathbf{u}_0, \mathbf{u}_1); \{r\})$ satisfies the following:

$$\begin{aligned}\mathbf{u}_0^{(r)} + s \cdot \mathbf{u}_1^{(r)} =& P^{(r)} \mu \cdot \left(\sum_{k=0}^{r-1} \frac{Q_{\ell_{\max}^{(r)}}}{q_k}, \ldots, \sum_{k=(d_{\max}^{(r)}-1)r}^{\ell_{\max}^{(r)}-1} \frac{Q_{\ell_{\max}^{(r)}}}{q_k} \right) \\ &+ \left(\sum_{k=0}^{r-1} e_k, \ldots, \sum_{k=(d_{\max}^{(r)}-1)r}^{\ell_{\max}^{(r)}-1} e_k \right) \pmod{Q_L}\end{aligned}$$

Let us represent the gadget vector and the noise parts as $\bar{\mathbf{g}} = (\bar{g}_0, \ldots, \bar{g}_{d_{\max}^{(r)}-1})$ and $\mathbf{e} = (\bar{e}_0, \ldots, \bar{e}_{d_{\max}^{(r)}-1})$, respectively. Then, during the computation of $\texttt{KS}(a; \{(\mathbf{u}_0^{(r)}, \mathbf{u}_1^{(r)})\})$, it satisfies that

$$\langle \mathbf{b}, (\bar{g}_0, \ldots, \bar{g}_{d-1}) \rangle = a \pmod{Q_\ell} \tag{3}$$

$$\mathbf{v}_0 + s \cdot \mathbf{v}_1 = P^{(r)} \mu \cdot (\bar{g}_0, \ldots, \bar{g}_{d-1}) + (\bar{e}_0, \ldots, \bar{e}_{d-1}) \pmod{P^{(r)} Q_\ell}$$

where Eq. (3) holds due to the observation in Sec. 3.2. Thus, (c_0, c_1) satisfies the followings:

$$\begin{aligned} c_0 + c_1 s &\approx \left\lfloor \frac{1}{P^{(r)}} \langle \mathbf{b}, \mathbf{v}_0 + s \cdot \mathbf{v}_1 \rangle \right\rceil & (4) \\ &= \left\lfloor \frac{1}{P^{(r)}} \left\langle \mathbf{b}, P^{(r)} \mu \cdot (\bar{g}_0, \ldots, \bar{g}_{d-1}) + (\bar{e}_0, \ldots, \bar{e}_{d-1}) \right\rangle \right\rceil \\ &= a\mu + \left\lfloor \frac{1}{P^{(r)}} \langle \mathbf{b}, (\bar{e}_0, \ldots, \bar{e}_{d-1}) \rangle \right\rceil \\ &\approx a\mu \pmod{Q_\ell} & (5) \end{aligned}$$

We note that Eq. (5) holds due to

$$\|\mathbf{b}\|_\infty \leq \frac{1}{2} \max_{0 \leq j < d} D_j^{(r)} \approx \frac{1}{2} P^{(r)}$$

since $D_j^{(r)}$ and $P^{(r)}$ are products of r primes, and we set each prime q_i to have a similar scale. Thus, key-switching keys are well-formed and the key-switching algorithm correctly works in our new framework.

Noise Analysis. Finally, we analyze a precise upper bound for noise occurred during our key-switching algorithm. We note that key-switching noise comes from Eq. (4) and (5). For Eq. (4), noise occurs due to rounding operations, thus it is bounded as follows

$$\begin{aligned} &\left\| \left\lfloor \frac{1}{P^{(r)}} \langle \mathbf{b}, \mathbf{v}_0 + s \cdot \mathbf{v}_1 \rangle \right\rceil - \left\lfloor \frac{1}{P^{(r)}} \langle \mathbf{b}, \mathbf{v}_0 \rangle \right\rceil - s \cdot \left\lfloor \frac{1}{P^{(r)}} \langle \mathbf{b}, \mathbf{v}_1 \rangle \right\rceil \right\|_\infty \\ &\leq \|e_{rd,1} + e_{rd,2} + s \cdot e_{rd,3}\|_\infty \leq 1 + \frac{N}{2} \|s\|_\infty \end{aligned}$$

where $e_{rd,i}$ corresponds to the rounding error of the i-terms in the above for $i = 1, 2, 3$. For Eq. (5), the noise is bounded as follows.

$$\left\| \left\lfloor \frac{1}{P^{(r)}} \langle \mathbf{b}, (\bar{e}_0, \ldots, \bar{e}_{d-1}) \rangle \right\rceil \right\|_\infty \leq \frac{1}{2} + \frac{NL\|\mathbf{e}\|_\infty}{2} \cdot \frac{\max_{0 \leq j < d_{\max}^{(r)}} D_j^{(r)}}{P^{(r)}}$$

Then, the total bound is as follows:

$$\frac{3}{2} + \frac{N}{2} \left(\|s\|_\infty + L\|\mathbf{e}\|_\infty \cdot \frac{\max_{0 \leq j < d_{\max}^{(r)}} D_j^{(r)}}{P^{(r)}} \right)$$

We again note that an upper bound for key-switching noise is small since $P^{(r)} \approx \max_{0 \le j < d^{(r)}_{\max}} D_j$ holds for all r.

5 Experiment Results

In this section, we present practical aspects of our new framework through experimental results. We first show how our framework optimizes key-switching performance at each level through benchmark data. Then, we provide several implications of our framework concerning real-world applications based on leveled HE schemes.

Our implementation is based on version 3.0.4 of the Lattigo library [26], which implements the CKKS and BFV schemes in a full RNS manner. We conduct all benchmarks on a laptop machine equipped with an Apple M2 CPU and 16GB of RAM, and all benchmarks are performed using a single thread. The source code is available at `https://github.com/SNUCP/level-aware-ksw`.

5.1 Parameter Settings

We present parameter settings for our implementation. Basically, we follow the parameter settings of [2], which provides standard parameter sets for lattice-based HE schemes. To be precise, the key distribution χ samples each coefficient from $\{0, \pm 1\}$ with probability $1/3$ for each element. The error distribution ψ is set to the discrete Gaussian D_σ with $\sigma = 3.2$. We use the ring dimension $N = 2^{16}$ which provides large enough depths for evaluating complex arithmetic circuits. We set $\lceil \log Q_L \rceil = 1760$ which gives 128-bit security with respect the forementioned RLWE parameters N, χ, ψ. We use the primes q_i's satisfying $q_i = 1 \pmod{2N}$ so that they can have a $(2N)$-th root of unity for NTT. Additionally, we fixed the total number of prime factors as $L = 40$ and set the size of prime factors to be approximately 2^{44} so that they have roughly the same scale. Table 2 summarizes our parameter setting.

N	$\lceil \log Q_L \rceil$	L	$\lceil \log q_i \rceil$
2^{16}	1760	40	44

Table 2: Parameters used in our experiments. L stands for the number of prime factors for Q.

5.2 Benchmark Results

To identify performance enhancement from our new framework, we measure the elapsed time of key-switching operations at each level using both the new and previous framework. Each key-switching operation is performed on random ciphertexts at levels $1 \le \ell \le 39$. In the benchmark, we use various digit lengths $r = 1, 2, 4, 8,$ and 16 to demonstrate the effect of digit lengths on key-switching performance. The result is presented in Table 3 and Fig. 1.

In the previous framework, once a digit length is determined at the setup phase, it affects the maximum ciphertext level and key-switching performance at all levels. For the maximum ciphertext level, it is determined as $L - r$, so small r values are preferred. However, when it comes to the performance of key-switching, a larger r usually gives better results at high levels, as shown in Table 3 when $\ell \ge 28$. This is because the complexity of both NTT and Hadamard product involves an $O(d\ell)$ factor, as presented in Table 1, where $d = \lceil \ell / r \rceil$. On the contrary, in the case of low d values (*i.e.,* the ratio between ℓ and r is small), a larger r may degrade key-switching performance since the $O(r)$ factor in NTT has a significant impact in such cases, as shown in Table 3 when $\ell \le 12$ or $r = 16$.

Hence, setting the proper r is in a trade-off relation with various factors, and it is crucial to find the right balance between these factors in the previous framework. For example, when $r = 1$, we can achieve

ℓ	Previous					Ours
	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 16$	
39	**962**		-	-	-	961
38	926	**691**	-	-	-	694
37	871	**665**		-	-	664
36	825	623	**417**	-	-	417
35	797	596	**404**	-	-	403
34	741	557	**390**	-	-	390
33	699	531	**370**	-	-	371
32	660	494	338	**266**	-	265
28	513	376	267	**230**	-	230
24	384	281	204	**174**	202	173
20	274	203	150	**145**	172	145
16	183	140	107	**102**	114	102
12	110	88	**72**	79	96	71
8	55	48	**43**	48	73	43
4	**19**	20	21	32	52	19

Table 3: Benchmark results of previous and our key-switching algorithm. ℓ denotes the level of ciphertext. We measure performance of previous key-switching operations with $r = 1, 2, 4, 8, 16$. Ours stands for our algorithm with optimal r chosen from those values. Unit of all operations is milliseconds per operation.

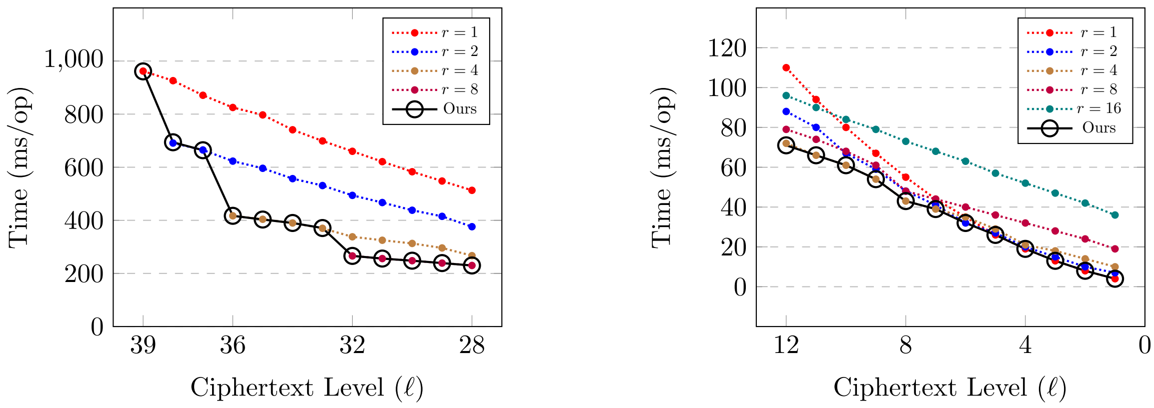

Fig. 1: Performance of key-switching operation when $N = 2^{16}$. $r = 1, 2, 4, 8, 16$ stands for the previous method with each r, and Ours stands for our new method.

the highest ciphertext level, but key-switching performance is slower than other r values in most levels. On the contrary, when $r = 8$, we gain optimal performance in most levels, but we need to sacrifice 7 levels, which is about 18% of available levels. With $r = 4$, we obtain a somewhat balanced result between the $r = 1$ and $r = 8$ cases at high levels, and it provides the best performance at low levels $\ell \leq 12$.

In the level-aware key-switching framework, we are free from such concerns, and we can always take advantage of each r value by constructing an index map I according to the benchmark results. As shown in Table 3 and Fig. 1, our framework matches the best-performing r in each level from previous key-switching framework. Hence, we can perform key-switching operations from $\ell = 39$, and the key-switching performance gets accelerated as level goes down since we can utilize more digit lengths. For example, when $\ell = 32$, our framework achieves about 2.5x speed-up compared to the previous framework with $r = 1$. When we reach low levels ($\ell \leq 12$), we start to reduce digit lengths as small r values give better performance in this case. As another example, when $\ell = 4$, our framework achieves about 1.7x speed-up compared to the previous framework with $r = 8$. Therefore, our framework always provides optimal key-switching performance across all levels by dynamically adjusting digit lengths.

Degree	Evaluation Time (s)		Speedup
	Previous	Ours	
8	7.77	5.67	1.37x
16	19.56	13.55	1.44x
32	48.74	31.28	1.55x

Table 4: Benchmark results for polynomial evaluation

To assess the impact of our framework in real-world applications, we measure the evaluation time for random polynomials in Table 4. For the previous method, we fix r at 1, while for ours, we use the optimal r values presented in Table 3 at each level. Both methods use input ciphertexts with level $\ell = 39$. For polynomials with degrees ranging from 8 to 32, our method achieves 1.37 to 1.55 times speed-up. Also, as the degree of the polynomial increases, the speed-up factor improves. This is because the performance improvement accelerates as the level decreases when $\ell = 32 \sim 39$, as presented in Fig. 1. Thus, our framework can provide a significant speed-up when evaluating large-depth circuits, which often includes the evaluation of large-degree polynomials.

Meanwhile, our framework requires to precompute several key-switching keys to enjoy such performance enhancement. To analyze overheads from preprocessing for keys, we measure the elapsed time for the key expansion algorithm in Table 5. Our experiments show that the extra runtime induced from the key expansion algorithm is insignificant compared to the complexity improvements in key-switching operations at high levels, such as $\ell = 28 \sim 39$. Additionally, this preprocessing is performed only once throughout the entire evaluation phase. Hence, we conclude that our framework still offers better performance compared to the previous one, even when considering the additional cost from preprocessing for keys.

r	Expansion Time (ms)	Key Size (GB)
1	-	1.63
2	77	0.79
4	73	0.37
8	64	0.16
16	48	0.08

Table 5: Benchmark results for key expansion algorithm

5.3 Implications

We end this section by providing several implications of our framework. As noted before, our framework has a significant advantage when evaluating large-depth arithmetic circuits, as it allows for dynamic adjustment of proper digit lengths at each level. This flexibility provides enhanced performance for various applications of HE. For instance, our framework can be utilized to accelerate the homomorphic evaluation of the CKKS bootstrapping circuit [8, 25, 24, 4], comparison function [11, 12], logistic regression [19, 21], and convolutional neural networks [22, 23], all of which involve evaluating large-depth circuits. By applying our level-aware key-switching framework to these computations, one can achieve enhanced performance, making homomorphic encryption more practical for real-world applications with complex computations and deep circuits.

One can assess our contribution in terms of saving communication costs since our framework enables an evaluator to derive key-switching keys with various digits, which is equivalent to the scenario where a key generator generates these key-switching keys and sends them to evaluators on demand. Therefore, our framework eliminates the need for sending additional key-switching keys, resulting in communication cost savings. Furthermore, in our framework, an evaluator does not need to maintain expanded keys throughout the entire evaluation process, as they can be regenerated at any time. This also provides an advantage in space utilization compared to the previous framework.

To be precise, for expanding key-switching keys to digits $r = 2, 4, 8, 16$, it takes about 262 milliseconds, and the total blow-up is about 1.4 gigabytes (see Table 5). As a comparison, if we assume a key generator transmits these additional keys with a speed of 1GBit/s, it takes 11.2 seconds, which is about 43 times slower. This effect gets magnified when the number of initial key-switching keys becomes larger. For example, in the Lattigo library [26], a key generator needs to send 48 key-switching (automorphism) keys to an evaluator to perform CKKS bootstrapping. With our framework, we can save communication costs of transmitting about 70 gigabytes in this case.

6 Conclusion and Future Work

In this paper, we present a new key-switching framework that allows choosing the optimal digit length at each level. With the previous framework [17, 18], one has to fix the digit length during the setup phase, which determined the maximum available level and key-switching performance at all levels. Moreover, key-switching performance and available level is in a trade-off relationship. In our framework, by exploiting the structure of the RNS-based gadget vector, we eliminate this trade-off, allowing one to convert digit lengths of key-switching keys via simple additions. As a result, our framework achieves the optimal key-switching performance at each level without compromising available levels.

One direction for further work is to apply our framework to a recent study by Kim et al. [20], which improves the asymptotic complexity of the key-switching algorithm. Since their work also utilizes RNS-based gadget toolkits, it is compatible with our framework. Moreover, as their optimization is orthogonal to ours, we can potentially achieve even greater efficiency in key-switching operations by leveraging the advantages of both works.

References

1. Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., et al.: Openfhe: Open-source fully homomorphic encryption library. In: Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 53–63 (2022)
2. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption security standard. Tech. rep., HomomorphicEncryption.org, Toronto, Canada (November 2018)
3. 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 (2016)

4. 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)
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
7. Chen, H., Laine, K., Player, R.: Simple encrypted arithmetic library-seal v2. 1. In: Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21. pp. 3–18. Springer (2017)
8. 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)
9. 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)
10. 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)
11. 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)
12. 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)
13. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
14. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
15. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
16. 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)
17. Han, K., Ki, D.: Better bootstrapping for approximate homomorphic encryption. In: Cryptographers' Track at the RSA Conference. pp. 364–390. Springer (2020)
18. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 608–639. Springer (2021)
19. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.H.: Logistic regression model training based on the approximate homomorphic encryption. BMC medical genomics **11**(4), 23–31 (2018)
20. Kim, M., Lee, D., Seo, J., Song, Y.: Accelerating he operations from key decomposition technique. In: Annual International Cryptology Conference. p. 70–92. Springer (2023)
21. Kim, M., Song, Y., Wang, S., Xia, Y., Jiang, X., et al.: Secure logistic regression based on homomorphic encryption: Design and evaluation. JMIR medical informatics **6**(2) (2018)
22. Lee, E., Lee, J.W., Lee, J., Kim, Y.S., Kim, Y., No, J.S., Choi, W.: Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In: International Conference on Machine Learning. pp. 12403–12422. PMLR (2022)
23. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., et al.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. IEEE Access **10**, 30039–30054 (2022)
24. 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: Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40. pp. 618–647. Springer (2021)
25. 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)
26. Mouchet, C.V., Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Lattigo: A multiparty homomorphic encryption library in go. In: Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography. pp. 64–70 (2020)