Efficient Key-Switching for Word-Type FHE and GPU Acceleration

Shutong Jin, Zhen Gu, Guangyan Li, Donglong Chen, Çetin Kaya Koç, Ray C. C. Cheung and Wangchen Dai

***Abstract*—Speed efficiency, memory optimization, and quantum resistance are essential for safeguarding the performance and security of cloud computing environments. Fully Homomorphic Encryption (FHE) addresses this need by enabling computations on encrypted data without requiring decryption, thereby maintaining data privacy. Additionally, lattice-based FHE is quantum secure, providing defense against potential quantum computer attacks. However, the performance of current FHE schemes remains unsatisfactory, largely because of the length of the operands and the computational expense associated with several resource-intensive operations. Among these operations, key-switching is one of the most demanding processes because it involves complex arithmetic operations necessary to conduct computations in a larger cyclotomic ring.**

In this research, we introduce a novel algorithm that achieves linear complexity in the Number Theoretic Transform (NTT) for key-switching. This algorithm offers efficiency comparable to the state-of-the-art while being significantly simpler and consumes less GPU memory. Notably, it reduces space consumption by up to 95%, making it highly friendly for GPU memory. By optimizing GPU performance, our implementation achieves up to a 2.0$\times$ speedup compared to both the baseline approach and the current state-of-the-art methods. This algorithm effectively balances simplicity and performance, thereby enhancing cryptographic computations on modern hardware platforms and paving the way to more practical and efficient FHE implementations in cloud computing environments.

***Index Terms*—Key-Switching, FHE, GPU implementation, BFV, BGV, CKKS**

I. Introduction

Fully-Homomorphic Encryption (FHE) is a cryptographic system that enables computations on encrypted data without needing to decrypt it first. This capability is essential for preserving data privacy across a wide range of applications, as it enables secure data processing without exposing sensitive information. FHE is particularly important and promising, as it naturally addresses the need for privacy-preserving solutions in cloud computing environments. Its ability to perform secure computations on encrypted data makes it an ideal candidate for applications where data privacy is paramount. On the other hand, FHE is grounded in lattice cryptography. Specifically, the security of contemporary FHE schemes is based on the hardness of lattice-based problems, such as the Learning With Errors (LWE) problem. This reliance provides a robust defense against quantum computer attacks, as these problems are believed to be resistant to quantum algorithms like Shor's algorithm, which threatens traditional cryptographic systems such as RSA and ECC. Consequently, lattice-based FHE schemes offer a promising pathway for maintaining security in the advent of quantum computing.

The initial application of FHE to Privacy-Preserving Machine Learning (PPML) was effectively demonstrated through its use in securing decision tree models, as illustrated by the work of Khedr et al., 2015 [28]. This foundational trial paved the way for further extensions of FHE to more complex machine learning models, including neural networks. Subsequent research by Aslett et al. (2015) [8], Chabanne et al. (2017) [13], and others [6, 7, 15, 15, 27] expanded the application of FHE to neural networks, showcasing its potential in privacy-preserving machine learning. These advancements underscore the transformative impact of FHE in enabling secure and private data processing in modern computational environments. The promising future of FHE lies in its ability to secure data in multi-party and sensitive scenarios, such as finance and medical applications, where data privacy and integrity are crucial.

Key-switching is a fundamental operation in FHE that enables the transformation of ciphertexts between different keys or moduli, which is essential for maintaining the correctness of computations as the noise level increases. It is a computationally intensive arithmetic operation in FHE. However, it is also integral to several critical processes, such as bootstrapping, ciphertext rotation, and relinearization. The computational expense of key-switching arises from the need to perform inner product computations on a different cyclotomic ring. To prevent further noise introduction after switching keys, the inner product must be computed in a larger modulus space, denoted $\mathbf{Q}_\ell \mathbf{P}$, with a newly introduced Residue Number System (RNS) base $\mathbf{P}$ that is specifically used in key-switching. This requirement necessitates expensive arithmetic operations, including base conversion and (Inverse) Number Theoretic Transforms (NTTs). Consequently, improvements in key-switching can lead to substantial improvements in the overall efficiency of FHE schemes. Therefore, key-switching becomes one of the main bottlenecks in terms of computational complexity and efficiency of the overall FHE schemes.

In our research, we are particularly focused on enhancing key-switching because the efficiency of current FHE schemes is limited by this process. Each instance of ciphertext multiplication requires a key-switching operation from $\mathbf{s}^2$ to $\mathbf{s}$. Similarly, every instance of ciphertext rotation requires an appropriate adjustment of the secret key. Furthermore, efficient key-switching is crucial for enabling more complex operations, such as bootstrapping, which is essential for refreshing ciphertexts and managing noise growth. *This is particularly important in applications that involve deep computational circuits, where noise accumulation can otherwise render ciphertexts undecipherable.*

Given the widespread necessity of key-switching in FHE, optimizing this process can significantly reduce the overhead associated with these transformations, thereby enhancing the overall performance of the encryption scheme. By focusing on improving key-switching, we aim to address one of the primary bottlenecks in FHE, ultimately making these schemes more practical and efficient for real-world applications.

A. Related Works

a) Development of FHE schemes: The foundational work, Gentry (2009) [20] established the theoretical framework for FHE, marking a significant milestone in the field of cryptography. Since then, lattice-based methods have emerged as the preferred approach for constructing homomorphic encryption schemes, because of their efficiency and robust security properties. Among these, schemes based on the LWE problem [9][11][12][22] and its ring variant (RLWE) [10][33][21] have gained significant traction, often referred to as word-type FHE schemes. These LWE/RLWE-based schemes are particularly favored for their ability to support Single Instruction, Multiple Data (SIMD) operations and batching. This capability is crucial because it allows multiple plaintexts to be processed simultaneously, significantly improving the throughput of homomorphic computations. This is especially important given that the speed of FHE implementations has historically been a limiting factor in their practical use.

b) Research in Key-switching: The Gentry, Halevi, and Smart (GHS) key-switching method [21] requires only a linear number of NTTs and is recognized for its efficiency. However, the GHS technique requires either doubling the dimension N or halving the size of the modulus Q to maintain security, which can be a limitation in certain scenarios. In contrast, the Brakerski and Vaikuntanathan (BV) key-switching technique [10], although more straightforward in its application, involves a quadratic number of NTTs. Hybrid key-switching [26] effectively combines the digit decomposition technique from BV and the larger modulus approach from GHS to achieve an optimal balance between noise growth and computational efficiency. Subsequent research has focused on refining these techniques to further enhance efficiency. For example, the work in [30] addresses the challenge of selecting an optimal decomposition strategy for hybrid key-switching. Furthermore, [31] introduces the concept of double decomposition to improve efficiency. This approach can be considered the current state-of-the-art, as it significantly reduces the computational burden associated with NTTs.

c) GPU Accelerated FHE: Given the support for SIMD operations in word-type FHE schemes, it is logical to consider using the computational power of GPUs for acceleration. GPUs are ideally suited for this task due to their capability for multithreaded computing. Previous research in this domain [2, 4, 5, 18, 23, 34, 35] has shown that exploiting the parallel processing power of GPUs can substantially accelerate FHE operations. Badawi et. al. (2020) [3] showed that multi-GPU could also be used to accelerate FHE schemes. The parallel processing capabilities of GPUs enable the simultaneous execution of numerous threads, which makes them particularly beneficial for handling repetitive and resource-intensive tasks inherent in FHE.

The open source community for Fully Homomorphic Encryption (FHE) features prominently popular libraries such as SEAL [14], HElib [25], and OpenFHE [1]. However, support for GPU acceleration within these libraries is not as well developed. The works [18], [36], and [29] are the representative GPU libraries available in the community. Our development is built upon *Phantom* [36], a high-performance GPU-accelerated library for FHE. *Phantom* specifically implements second-generation FHE schemes, including the RNS variants of BGV, BFV, and CKKS, which are designed to optimize computational efficiency and memory usage. Phantom provides robust support for batch processing and SIMD operations with easy-to-use API and utility tools. The arithmetic implementations reach the state-of-the-art for FHE-GPU.

B. Our Contribution

In this work, we introduce a novel key-switching method. We show that this simple, yet effective, key-switching approach achieves performance comparable to the best results reported by [31], while significantly enhancing space efficiency. Furthermore, by substantially reducing space consumption, this method becomes highly suitable for GPU implementation. Our contributions are detailed as follows.

- **Linear-Keyswitching Method:** We introduce a novel method termed linear-keyswitching, which addresses the substantial complexity associated with performing inner products in large modulus spaces. Our analysis reveals that this method reduces the complexity of unit NTT operations from $O(\ell^2)$ to $O(\ell)$. Compared to the current state-of-the-art, it offers superior time and space efficiency while alleviating concerns related to choosing parameters.
- **GPU Implementation and Optimization:** We have implemented our algorithms within a GPU library with CUDA support. Due to the simplicity of our approach, we incorporated kernel-fusing and enhancements in the base-conversion process, resulting in significant speed performance improvements on the target device. This method streamlines data handling and arithmetic operations, enabling more efficient execution on GPU architectures. It also highlights the potential for further optimization based on hardware capabilities and parameter selection.
- **Review and Analysis of Current State-of-the-Art on GPU:** By implementing the complete version of the double decomposition approach from [31], we identified several challenges in adapting this method for GPU development, including complex parameter selection and high GPU memory demands, which complicate its realization. Our analysis underscores the difficulties and limitations of adapting [31] for GPU implementation, offering valuable insights for future research.
- **Comprehensive Benchmarking:** To validate the efficiency of our method, we conducted extensive comparisons among the GPU implementation of our method, the double decomposition, and the baseline approach, covering all word-type schemes and a set of public parameters. The results demonstrate that our method achieves efficiency in a GPU context that is comparable to, if not superior to, existing solutions. Remarkably, while maintaining runtime efficiency, our method reduces memory requirements by over 95%. This optimization results in a more developer-friendly and memory-efficient

TABLE I: List of Symbols

Symbol	Description
α	Number of special modulus
ℓ	Current level of modulus chain
β	Number of relinKeys involved in key-switching $= \lceil \ell/\alpha \rceil$
$\mathbf{P}$	Mod-up RNS Base $p_0 \cdot \ldots p_\alpha$
$\mathbf{Q}_\ell$	RNS Base $q_0 \cdot \ldots q_\ell$
$\mathbf{Q}_\ell\mathbf{P}$	RNS Base $q_0 \cdot \ldots q_\ell \cdot p_0 \cdot \ldots p_\alpha$
$\mathbf{Q}$	Full mdulus chain $q_0 \cdot \ldots q_{\ell\max}$
$m_0 \cdot m_1$	RNS base of two primes m_0 and m_1
$\lvert\mathbf{Q}_\ell\rvert$	number of primes in $\mathbf{Q}_\ell$
$\lvert\mathbf{Q}_\ell\mathbf{P}\rvert$	number of primes in $\mathbf{Q}_\ell\mathbf{P}$
$\lvert\mathbf{Q}\rvert$	number of primes in $\mathbf{Q}$
$\lvert\mathbf{P}\rvert$	number of primes in $\mathbf{P}$

implementation, facilitating easier integration and deployment in practical applications.

Our contributions significantly advance the research on GPU-accelerated FHE, offering new methodologies and insights that enhance both theoretical understanding and practical application.

II. Preliminaries

The Learning With Errors (LWE) problem is based on the difficulty of solving noisy linear equations over finite fields. Given an integer n and the modulus q, LWE involves finding a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ from sample $(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, where $\mathbf{a}$ is uniformly sampled and b is computed as $b = \langle \mathbf{a}, \mathbf{s} \rangle + \mathbf{e} \pmod{q}$. $\mathbf{e}$ is introduced as a small error from a distribution like a discrete Gaussian.

The Ring Learning With Errors (RLWE) problem extends LWE to polynomial rings. Defined over $\mathcal{R}_q = \mathbb{Z}_q[x]/(f(x))$, RLWE involves finding a secret polynomial $s(x) \in \mathcal{R}_q$ from samples $(a(x), b(x)) \in \mathcal{R}_q^2$, where $a(x)$ is uniformly sampled and $b(x) = a(x) \cdot s(x) + e(x) \pmod{q}$, with $e(x)$ having small error coefficients.

Consider $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ be a ring of polynomials with integer coefficients modulo $X^N + 1$, where N is a power of two. A plaintext μ is encrypted as a ciphertext $\mathbf{c} = [\mathbf{c_0}, \mathbf{c_1}]$.The description holds $\mathbf{c_0} + \mathbf{c_1} \cdot \mathbf{s} = \mathbf{m} + \mathbf{e}$, where $\mathbf{s}$ is the secret key, $\mathbf{e}$ is the error term, and $\mathbf{m}$ is the plaintext.

A. Word-Type FHE Schemes

The word-type FHE Schemes represents a significant evolution from Gentry's initial 2009 blueprint [20], which was based on the Closest Vector Problem (CVP) on ideal lattices. Instead, these newer schemes shift focus to the LWE and Ring Learning With Errors RLWE problems, which are not required to be on ideal lattices.

Significantly, the Brakerski-Vaikuntanathan (BV) scheme in 2014 [11] can be seen to be the foundation of the word-type schemes. The initial proposal was based on the LWE problem, which is NP-hard. Meanwhile, this scheme introduced the concept of relinearization, a technique designed to address the complexity issues inherent in Gentry's original scheme [20] by reducing the dimensionality of ciphertexts. Relinearization effectively transforms quadratic terms into linear terms, thereby mitigating the dimension explosion that occurs during homomorphic multiplications.

The Brakerski-Fan-Vercauteren (BFV) [19] scheme (2012) represents a simplified adaptation of the BV scheme, transitioning from the LWE setting to the RLWE basis. This adaptation makes the scheme more practical for implementation while retaining the core advantages of its predecessors.

Building on the BV scheme, Gentry, Brakerski, and Vaikuntanathan developed the BGV scheme in 2014 [9]. This scheme further refined the concept of relinearization by incorporating key-switching, which facilitates various forms of complexity reduction.

The Cheon-Kim-Kim-Song (CKKS) scheme [17], also based on the BV scheme and RLWE, introduces a novel approach utilizing the complex plane. This allows for the representation of floating-point numbers and the use of scientific notation, which is particularly beneficial in applications involving neural networks where precise floating-point computations are essential. The CKKS scheme's support for floating-point arithmetic enhances its applicability in modern computational tasks.

Below, we provide a comprehensive description of the functionalities of the three major schemes of the word-type FHE: BGV, BFV, and CKKS.

- **Setup:** For a given security parameter λ, select appropriate public parameters $pp = (N, t, Q, P, \chi_{\text{key}}, \chi_{\text{err}})$, including a a ring dimension N, a plaintext modulus $|\mathbf{T}|$, a ciphertext modulus Q, and distributions χ_{key} and χ_{err}.
- **Key Generation:** Generate a secret key sk and a public key $\mathbf{pk}$. Sample $\mathbf{s} \leftarrow \chi_{\text{key}}$ and set $sk = \mathbf{s}$. Sample a random element $\mathbf{a} \in R_Q$ and an error term $\mathbf{e} \leftarrow \chi_{\text{err}}$. Set $\mathbf{pk} = (\mathbf{a}, -\mathbf{as} + \mathbf{e}$. Generate relinearization keys $\mathbf{pk_{relin}}$ and automorphism keys $\mathbf{pk_{auto}}$.
- **Encryption:** Encrypt a plaintext $\mathbf{m}$ using the public key $\mathbf{pk} = (\mathbf{a}, -\mathbf{as} + \mathbf{e}) = (\mathbf{a}, \mathbf{b})$. Sample $\mathbf{r} \leftarrow \chi$ and $\mathbf{e_0}, \mathbf{e_1} \leftarrow \chi_{\text{err}}$. Calculate $Enc_{pk}(0) = [\mathbf{r} \cdot (\mathbf{a}, \mathbf{b}) + \mathbf{t} \cdot (\mathbf{e_0}, \mathbf{e_1})]_Q$. Output a ciphertext $\mathbf{ct} = (\mathbf{c_0}, \mathbf{c_1}) = [Enc_{pk}(0) + (\mathbf{m}^*, 0)]_Q$.
  - **BFV:** $\mathbf{m}^* = \lfloor \frac{Q}{t} \rceil \cdot [\mathbf{m}]_t$
  - **BGV:** $\mathbf{m}^* = [\mu \mathbf{m}]_t$, μ is the correction factor varying with the level
  - **CKKS:** $\mathbf{m}^* = \mathbf{m}$
- **Decryption:** Decrypt a ciphertext using the secret key.
  - **BFV:** $\mathbf{m} = \lfloor \frac{t}{Q} \cdot [\mathbf{c_0} + \mathbf{c_1} \cdot s]_Q \rfloor$
  - **BGV:** $\mathbf{m} = [\mu^{-1}[\mathbf{c_0} + \mathbf{c_1} \cdot s]_Q]$.
  - **CKKS:** $\mathbf{m} = [\mathbf{c_0} + \mathbf{c_1} \cdot s]_Q$
- **Homomorphic Operations:** Perform operations on ciphertexts.
  - **Addition:** Given two ciphertexts ct and ct', output $ct_{\text{add}} = ct + ct'$ for all schemes
  - **Multiplication:** Given two ciphertexts ct and ct' and a relinearization key $\mathbf{pk}_{\text{relin}}$, compute the product and relinearize to output ct_{mult} for all schemes
  - **Automorphism:** Apply an automorphism to a ciphertext using the automorphism key. Given a ciphertext ct and an automorphism key $\mathbf{pk_{auto}}$, output the transformed ciphertext ct_{auto}

B. Key-Switching

A switching key is a special public key designed to convert ciphertext encrypted under key $\mathbf{s_A}$ to ciphertext encrypted under a different key $\mathbf{s_B}$, while preserving the same plaintext segment. The definition of a switching key $\mathbf{pk}_{s_A \to s_B}$ follows:

$$\mathbf{pk}_{s_A \to s_B} = ([\mathbf{s_A} + \mathbf{a} \cdot \mathbf{s_B} + t\mathbf{e}]_Q, -\mathbf{a}) \in \mathcal{R}_Q^2$$

where $\mathbf{a}$ is a polynomial that is sampled uniformly in $\mathcal{R}_Q^2$, and $\mathbf{e} \leftarrow \chi_{\text{err}}$.

a) Relinearization: Following each ciphertext multiplication, the resulting ciphertext $\mathbf{c} = [\mathbf{c_0}, \mathbf{c_1}, \mathbf{c_2}]$ is encrypted under the basis $(\mathbf{1}, \mathbf{s}, \mathbf{s^2})$. The process of relinearization then transforms this ciphertext to find the corresponding components encrypted under the reduced basis $(\mathbf{1}, \mathbf{s})$. The key-switching keys used for relinearization are denoted as $\mathbf{pk}_{\text{relin}}$.

b) External Product: involves the computation of a linear combination of elements from a gadget vector $u = (u_i)_{0 \le i < d} \in R_Q^d$ with coefficients derived from the decomposition $\mathbf{b} = (b_i)_{0 \le i < d}$ of an element $\mathbf{a} \in \mathcal{R}_Q$. Formally, the external product can be expressed as:

$$\text{ExtProd}(a, u) = \sum_{i=0}^{d-1} b_i \cdot u_i \pmod{Q}$$

where b_i are small elements obtained from the gadget decomposition of a, such that $\mathbf{a} = \sum_{i=0}^{d-1} b_i \cdot g_i \pmod{Q}$, with $(g_i)_{0 \le i < d}$ being a fixed gadget basis over R_Q. This operation is crucial in key-switching techniques.

In our study, we define α as the number of co-primes in the mod-up base $\mathbf{P}$, and β as the number $\mathbf{pk}$ involved in the key-switching process. In works such as [26] and [21], the introduced base $\mathbf{P}$ may have multiple small moduli. For the purpose of simplifying our discussion, we focus primarily on the scenario where $\alpha = 1$, which implies that $\beta = \ell$, the level of the ciphertext.

In our implementation, we have simplified the key-switching process to accommodate both relinearization and other operations by utilizing $\mathbf{c_2}$ as the third term input to the key-switching module. This approach is particularly useful for unifying the treatment of different operations within the encryption scheme. Specifically, for scenarios that do not involve relinearization and require only two ciphertexts, we represent the inputs as $[\mathbf{c_0^*}, \mathbf{0}, \mathbf{c_1^*}]$. In this context, $\mathbf{c_2}$ effectively corresponds to the term $\mathbf{c_1^*}$, thus the second parameter serving as a placeholder to maintain consistency in the key-switching framework.

C. Modulus Switching

Modulus switching is a technique used to transition a ciphertext encrypted under a larger modulus Q to a smaller modulus Q', thus maintaining the noise level within acceptable limits and ensuring the correctness of decryption. This process is essential in homomorphic encryption schemes to manage noise growth during computations.

Mathematically, modulus switching is based on the concept of scaling. For a given ciphertext $ct = (\mathbf{c_0}, \mathbf{c_1})$ encrypted under modulus Q, the objective is to transform it into a new ciphertext $ct' = (\mathbf{c'_0}, \mathbf{c'_1})$ under modulus Q' while preserving the underlying plaintext. This transformation is typically achieved by scaling the ciphertext coefficients by the factor Q'/Q, followed by rounding to ensure that they fit within the new modulus Q'.

In our context, the term *level* and the symbol ℓ are used to denote the number of small moduli q_i currently in the modulus chain. The primes used for the ciphertext ring are expressed as $\mathbf{Q} = q_0 \cdots q_{\ell_{\max}}$. It is important to note that, unlike BGV and CKKS, the message in the BFV scheme is encrypted using the Most Significant Digits. Consequently, BFV is free from the need for modulus switching.

D. RNS Variants

Residue Number System (RNS) allows for the representation of large integers as tuples of smaller integers, particularly useful in computational contexts where parallel processing and modular arithmetic.

Consider a set of pairwise coprime integers $q_0, q_1, \ldots, q_{\ell-1}$. An integer a can be uniquely represented in RNS as a tuple $(\mathbf{a_0}, \mathbf{a_1}, \ldots, \mathbf{a_{\ell-1}})$, where $a_i = a \bmod q_i$. This representation is derived from the Chinese Remainder Theorem (CRT), which establishes an isomorphism between the ring $\mathcal{R}_\mathbf{Q} = \mathbb{Z}_\mathbf{Q}$ and the product of smaller rings $\mathcal{R}_{q_0} \times \mathcal{R}_{q_1} \times \cdots \times \mathcal{R}_{q_{\ell-1}}$, where $\mathbf{Q} = q_0 \cdot q_1 \cdots q_{\ell-1}$.

The RNS variant of the word-type scheme was initially proposed by Cheon et al. (2018) [16] and further improved by Halevi et al. (2018) [24], providing significant advancements in the efficiency of homomorphic encryption schemes by leveraging the parallelism inherent in RNS.

Base conversion is used in relinearization and involves converting a polynomial from a high-precision base to a lower-precision base. We define the operation of converting the base of a polynomial x from an input base to an output base as $\texttt{bconv}_{\mathbf{Q}\to\mathbf{P}}(x)$. At the implementation level, this work utilizes the HPS variant of base conversion, as described in [24]. In this context, the input base $\mathbf{Q}$ is expressed as the product $Q = \prod_{i=0}^{k} q_i$, where each q_i^* is calculated as $\mathbf{Q}/q_i$, and $\tilde{q}_i$ is defined as $[q_i^{*-1}]_{q_i}$. The output base $\mathbf{P} = \prod_{j=0}^{k'} r_j$. The base conversion function is given by:

$$\texttt{bconv}_{\mathbf{Q}\to\mathbf{P}}(x) = \left(\left[\sum_{i=0}^{k} [x_i \cdot \tilde{q}_i]_{q_i} \cdot q_i^* - v\mathbf{Q} \right]_{r_j} \right)_{j=0}^{k},$$

$$v = \left\lfloor \sum_{i=0}^{k} [x_i \cdot \tilde{q}_i / q_i] \right\rceil$$

E. GPU Programming

The GPU is designed as a multi-core processor with exceptional capabilities for parallel processing. This is achieved through the Compute Unified Device Architecture (CUDA), which provides a framework for efficiently utilizing GPU resources. In this architecture, the CPU initiates a **kernel**, which is executed simultaneously by numerous CUDA threads, allowing for massive parallelism.

The execution logic of a kernel can be effectively programmed using CUDA-like functions, which enable the concurrent execution of multiple kernels, thereby facilitating parallelism. Unlike traditional CPU programming, initiating a kernel on a GPU involves host-device communication, which introduces some latency. To mitigate this, an optimal approach for Single Instruction, Multiple Data (SIMD) operations is kernel fusion. By combining kernels from different modules, the overhead associated with kernel initialization is minimized, enhancing overall efficiency.

The parallel processing capability of GPU cards is significantly influenced by the number of CUDA cores they possess. For instance, the NVIDIA A100 demonstrates superior performance in executing arithmetic operations for polynomials of size 2^{16} due to its extensive parallel processing power. However, for smaller polynomial sizes, the grid may not be fully utilized, making the kernel initialization overhead more pronounced. Consequently, maximizing parallelism is desirable, especially when the target device specifications are not predetermined, as it allows for better resource utilization and performance optimization across various hardware configurations.

III. Linear-KeySwitching

For simplicity, all the algorithms discussed in this section are configured under the CKKS scheme, with parameters $\alpha = 1$ and $P = 1$, meaning there is only one introduced prime $p$in the mod-up ring $\mathcal{R}_{\mathbf{Q}_\ell \mathbf{P}}$.

A. Previous Approach

The previous key-switching algorithm, adapted from [26], encounters significant limitations when $\alpha > 1$, as the multiplication depth becomes highly restricted. For simplicity, this article primarily focuses on the case where $\alpha = 1$, which is illustrated in Algorithm 1, serving as the baseline approach for our study. Popular open-source libraries such as SEAL [14] incorporate similar algorithms. We have developed a GPU-accelerated version of this implementation, applicable to all word-type schemes, which we use as the baseline to assess the performance improvements and optimizations achieved by our new approach.

We observed that the primary issue with this approach lies in the mod-up module, specifically during the transformation of the input cipher from $\mathcal{R}_{\mathbf{Q}_\ell}$ to $\mathcal{R}_{\mathbf{Q}_\ell \mathbf{P}}$ (line 6 to line 7. This process involves a computationally intensive $O(\ell^2)$ times NTT for the ciphertext polynomial, which significantly contributes to the overall complexity. Moreover, the procedure involves $Q(\ell)$ times of base conversion from $\mathbf{Q}_\ell$ to $\mathbf{Q}_\ell \mathbf{P}$. The complexity of computation increases significantly as the level ℓ rises due to the quadratic growth in the number of NTT operations and base conversions. This complexity motivates us to address through the proposed method detailed in the following section.

Algorithm 1: Previous Approach: Key-Switching

```
Data: $\mathbf{c_2}$ on $\mathcal{R}_{\mathbf{Q}_\ell}$
Result: $\mathbf{c}'_2[c_0|c_1]$ on $\mathcal{R}_{\mathbf{Q}_\ell}$
1  $c' \leftarrow$ Inv-NTT($\mathbf{c_2}$, $\mathbf{Q}_\ell$);
2  for $i \leftarrow 0$ to $\ell - 1$ do
3      ibase $\leftarrow [q_i]$;
4      obase $\leftarrow \mathbf{Q}_\ell \mathbf{P}$ ;
5      $c'_i \leftarrow$ part $c'$ on $[q_{\alpha \cdot i}, q_{\alpha \cdot i + \alpha}]$;
6      $\mathbf{b_i} \leftarrow$BaseConversion($c'_i$, ibase, obase);
7      $\mathbf{b_i} \leftarrow$NTT($\mathbf{b_i}, \mathbf{Q}_\ell \mathbf{P}$);
8  end
9  for $\leftarrow 0$ to $\ell$ do
10     $[\mathbf{cx_i}]_{\mathbf{Q}_\ell \mathbf{P}} += \mathbf{b_i} \cdot [\mathbf{pk}]_i$;
11 end
12 for $i \leftarrow 0$ to $\ell + 1$ do
13     $\mathbf{cx}_j \leftarrow$ Inv-NTT($\mathbf{cx}_j, \mathbf{Q}_\ell \mathbf{P}$);
14 end
15 $\Delta[c_0] \leftarrow [\mathbf{cx}[c_0]]_p$;
16 $\Delta[c_1] \leftarrow [\mathbf{cx}[c_1]]_p$;
17 for $i \leftarrow 0$ to $\ell - 1$ do
18     $\Delta[c_0] \leftarrow [\Delta[c_0]]_{\mathbf{Q}_\ell}$ ;
19     $\Delta[c_1] \leftarrow [\Delta[c_1]]_{\mathbf{Q}_\ell}$ ;
20 end
21 for $i \leftarrow 0$ to $\ell$ do
22     $\mathbf{c}'_2[c_0|c_1] \leftarrow$
         ModDownKernel($\boldsymbol{cx}[c_0|c_1], \Delta[c_0|c_1]$) ;
23     $\mathbf{c}'_2[c_0|c_1] \leftarrow$ NTT($\boldsymbol{c}'_2[c_0|c_1], \mathbf{Q}_\ell$) ;
24 end
```

B. Our Method with Linear NTT Complexity

Recall that the ultimate objective is to compute the external product of $\mathbf{c_2}$ on $\mathcal{R}_{Q_\ell}$ with $\mathbf{pk_0}[c_0|c_1], \mathbf{pk_1}[c_0|c_1], \ldots, \mathbf{pk_{\beta-1}}[c_0|c_1]$ on $\mathcal{R}^2_{\mathbf{Q}_\ell \mathbf{P}}$. Traditionally, this process involved converting the base from $\mathbf{Q}_\ell$ to $\mathbf{Q}_\ell \mathbf{P}$ and performing the inner product in the NTT space of $\mathbf{Q}_\ell \mathbf{P}$. This approach can be computationally intensive due to the potentially large size of the RNS base, particularly when ℓ is substantial.

To address this complexity, we propose a novel method that circumvents the need for full base conversion from q_i to the entire RNS base of $\mathbf{Q}_\ell \mathbf{P}$. Instead, we explore the possibility of treating the polynomial of $[\mathbf{c_2}]_{q_i}$ as on $\mathcal{R}$ and manipulate the values itself for polynomial multiplication, where $q_i \in \mathbf{Q}_\ell$.

Consider an element $a \in \mathcal{R}$, and two co-prime integers form a new RNS base $[m_0, m_1]$. The relationship is expressed by the following equation:

$$x = [a \cdot m_1^{-1}]_{m_0} \cdot m_1 + [a \cdot m_0^{-1}]_{m_1} \cdot m_0$$

This equation can be used to demonstrate that

$$[a]_{m_0 m_1} = x$$

provided that $x < m_0 \cdot m_1$. This result is straightforward to prove due to the properties of the Chinese Remainder Theorem.

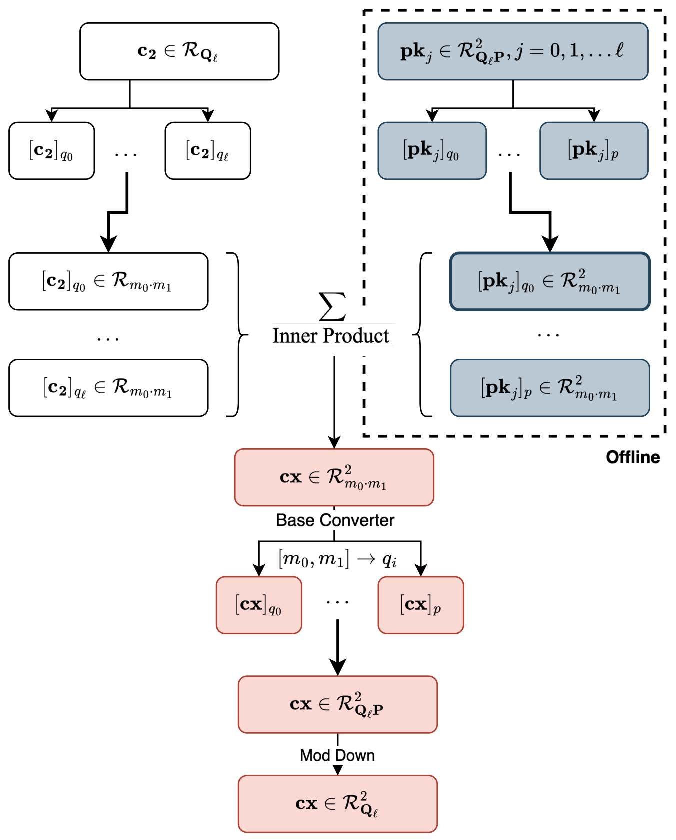

$\mathbf{c_2} \in \mathcal{R}_{\mathbf{Q}_\ell}$
$[\mathbf{c_2}]_{q_0}$
$\dots$
$[\mathbf{c_2}]_{q_\ell}$
$[\mathbf{c_2}]_{q_0} \in \mathcal{R}_{m_0 \cdot m_1}$
$\dots$
$[\mathbf{c_2}]_{q_\ell} \in \mathcal{R}_{m_0 \cdot m_1}$
$\sum$
Inner Product
$\mathbf{pk}_j \in \mathcal{R}^2_{\mathbf{Q}_\ell \mathbf{P}}, j = 0, 1, \dots \ell$
$[\mathbf{pk}_j]_{q_0}$
$\dots$
$[\mathbf{pk}_j]_p$
$[\mathbf{pk}_j]_{q_0} \in \mathcal{R}^2_{m_0 \cdot m_1}$
$\dots$
$[\mathbf{pk}_j]_p \in \mathcal{R}^2_{m_0 \cdot m_1}$
Offline
$\mathbf{cx} \in \mathcal{R}^2_{m_0 \cdot m_1}$
Base Converter
$[m_0, m_1] \to q_i$
$[\mathbf{cx}]_{q_0}$
$\dots$
$[\mathbf{cx}]_p$
$\mathbf{cx} \in \mathcal{R}^2_{\mathbf{Q}_\ell \mathbf{P}}$
Mod Down
$\mathbf{cx} \in \mathcal{R}^2_{\mathbf{Q}_\ell}$

Fig. 1: Overview of our proposed algorithm

TABLE II: Time complexity analysis of our approach

	(Inv) NTT	Hadamard Product
[26]	BGV: $\ell^2 + 7\ell + 2$ CKKS: $\ell \cdot (\ell + 1) + 5\ell$ BFV: $(\ell + 2) \cdot (\ell + 1)$	$2\ell \cdot (\ell + 1)$
Ours	BGV/CKKS: $8\ell + 4$ BFV: $6\ell + 4$	$4\ell \cdot (\ell + 1)$

Given the constraint that x is bounded, during polynomial multiplication, the sum of coefficients must not exceed $m_0 \cdot m_1$. This requirement ensures that the intermediate modulus can accommodate numbers of size $\max(q_i)^2 \times N$. Typically, q_i is less than 60 bits, leading to a maximum product width of $60 + 60 + 16$ bits. This width slightly exceeds 128 bits, making it impractical to use a single 128-bit modulus directly. Consequently, we have opted to utilize two 64-bit moduli for intermediate modulus operations. This choice is motivated by the fact that, although direct computation with a 128-bit modulus might be straightforward, employing two 64-bit moduli offers advantages in resource management and hardware implementation efficiency.

Algorithm 2 outlines the preprocessing steps for generating the key-switch key, while Algorithm 3 provides the pseudo-code for the online calculations of the proposed method within a CKKS framework. For details of the implementation of algorithms related to other schemes, refer to Section V.

The new methodology significantly the unit NTT operations complexity to $O(\ell)$, as demonstrated in Table II.

Algorithm 2: Linear-Keyswitching: $\mathbf{pk}_{\text{relin}}$ Processing

```
Data: $\mathbf{pk_0}[c_0|c_1], \mathbf{pk_1}[c_0|c_1], ..., \mathbf{pk_{\ell-1}}[c_0|c_1]$ on $\mathcal{R}^{\ell \times 2}_{\mathbf{Q}_\ell \mathbf{P}}$
Result: $\mathbf{v_{0,0}}[c_0|c_1], \mathbf{v_{0,1}}[c_0|c_1], ..., \mathbf{v_{\ell-1,\ell}}[c_0|c_1]$ on $\mathcal{R}^{\ell \times (\ell+1) \times 2}_{m_0 \cdot m_1}$
1  for $i \leftarrow 0$ to $\ell - 1$ do
2      for $j \leftarrow 0$ to $\ell$ do
3          $[\mathbf{v_{i,j}}]_{m_0} \leftarrow [\mathbf{pk}_j]_{q_i}$ ;
4          $[\mathbf{v_{i,j}}]_{m_1} \leftarrow [\mathbf{pk}_j]_{q_i}$ ;
5          $\mathbf{v_{i,j}} \leftarrow$NTT($\mathbf{v_{i,j}}, [m_0, m_1]$);
6      end
7  end
```

Algorithm 3: Linear-Keyswitching: Key-Switching

```
Data: $\mathbf{c_2}$ on $\mathcal{R}_{\mathbf{Q}_\ell}$
Result: $\mathbf{c}'_2[c_0|c_1]$ on $\mathcal{R}_{\mathbf{Q}_\ell \mathbf{P}}$
1  $c' \leftarrow$Inv-NTT($\mathbf{c_2}$, $\mathbf{Q}_\ell$);
2  for $i \leftarrow 0$ to $\ell - 1$ do
3      $[\mathbf{b_i}]_{m_0} \leftarrow [c']_{q_i}$ ;
4      $[\mathbf{b_i}]_{m_1} \leftarrow [c']_{q_i}$ ;
5      $\mathbf{b_i} \leftarrow$NTT($\mathbf{b_i}, [m_0, m_1]$);
6      for $j \leftarrow 0$ to $\ell$ do
7          $[\mathbf{cx_j}]_{m_0} += [\mathbf{b_i}]_{m_0} \cdot [\mathbf{v_{i,j}}]_{\mathbf{m_0}}$;
8          $[\mathbf{cx_j}]_{m_1} += [\mathbf{b_i}]_{m_1} \cdot [\mathbf{v_{i,j}}]_{\mathbf{m_1}}$;
9      end
10 end
11 for $j \leftarrow 0$ to $\ell$ do
12     $\mathbf{cx}_j \leftarrow$ Inv-NTT($\mathbf{cx}_j, [m_0, m_1]$);
13     $[\mathbf{cx}]_{q_j} \leftarrow$ BaseConversion($\boldsymbol{cx}_j, [m_0, m_1], q_j$);
14 end
15 $\Delta[c_0] \leftarrow [\mathbf{cx}[c_0]]_p$;
16 $\Delta[c_1] \leftarrow [\mathbf{cx}[c_1]]_p$;
17 for $i \leftarrow 0$ to $\ell - 1$ do
18     $\Delta[c_0] \leftarrow [\Delta[c_0]]_{\mathbf{Q}_\ell}$ ;
19     $\Delta[c_1] \leftarrow [\Delta[c_1]]_{\mathbf{Q}_\ell}$ ;
20 end
21 for $i \leftarrow 0$ to $\ell$ do
22     $\mathbf{c}'_2[c_0|c_1] \leftarrow$
         ModDownKernel($\boldsymbol{cx}[c_0|c_1], \Delta[c_0|c_1]$) ;
23     $\mathbf{c}'_2[c_0|c_1] \leftarrow$ NTT($\boldsymbol{c}'_2[c_0|c_1], \mathbf{Q}_\ell$) ;
24 end
```

IV. The Double Decomposition Approach

The methodology proposed by Kim et al. (2023) [31] in Asiacrypt represents the current state-of-the-art for addressing the complexity of the NTT in key-switching operations. The main concept of this approach is to handle the decomposition of public keys (**pk**) more efficiently by leveraging a double decomposition strategy. Instead of decomposing the public keys directly and computing the inner product in a traditional manner, the method introduces a novel way of managing these operations in a double-decomposed form. This technique reduces the computational burden associated with NTT operations. By breaking down the problem into smaller, more manageable components, the method allows

TABLE III: List of symbols specified for double decomposition

Symbol	Description
r	Word width of key-decomposition
d	Number of key-decomposition $= \lceil Q_\ell P/r \rceil$
$\mathbf{T}$	Extended RNS Base for inner product
$\lvert\mathbf{T}\rvert$	Number of modulus in $\mathbf{T}$

TABLE IV: Corrected time complexity analysis of [31]

(Inv) NTT	Hadamard Product
BGV/CKKS: $\ell \cdot \lvert\mathbf{T}\rvert + 2 \cdot d \cdot \lvert\mathbf{T}\rvert + 3\ell$ BFV:$\ell \cdot \lvert\mathbf{T}\rvert + 2 \cdot d \cdot \lvert\mathbf{T}\rvert$	$2\ell \cdot d \cdot \lvert\mathbf{T}\rvert$

for a more efficient computation process that minimizes the overhead typically encountered with large moduli. The double decomposition method effectively decouples the complexity of the NTT from the size of the modulus, enabling a linear reduction in computational complexity from $O(\ell^2)$ to $O(\ell)$.

More specifically, the parameters for the second decomposition, as described in Table III, play a critical role in influencing the complexity of the NTT and Hadamard product operations within the algorithm. The time complexity analysis is shown in Table IV.

Although the article introduces promising advancements in key-switching, there are some inaccuracies and ambiguities in the original paper. For example, [32] highlighted a miscalculation in the complexity of the NTT and the Hadamard product, as well as a lack of consideration of the differences between various schemes. Table IV presents the revised complexity analysis, addressing these issues. In our work, we have validated the findings of [32] and implemented the corrected methodology in all relevant schemes. Furthermore, we provide a comprehensive step-by-step algorithm in the appendix, Algorithm 4-7. This ensures clarity and accuracy in the application of the method, allowing for more reliable and efficient implementation.

A. Performance Limitations and Challenges

a) Difficulty in Determining Optimal Values for r and d: The time complexity of the key-switching operation is a complex, non-linear equation involving both the NTT and Hadamard product computations. This complexity makes it challenging to use a deterministic method to find the optimal values r and d. In our implementation, we observed that the value of r which offers the best NTT complexity does not necessarily guarantee the best overall performance. This discrepancy arises because the computational cost is not solely dependent on NTT complexity but also on the interplay with Hadamard product operations and other factors. Meanwhile, base conversion involves floating-point arithmetic, which introduces further complexity that is not accounted for in the asymptotic complexity of NTT and Hadamard operations. Additionally, the number of kernel initializations can significantly affect performance. Frequent initializations can lead to increased overhead, thereby affecting the overall performance of the system.

Fig. 2: Theoretical Complexity Approximation vs. Experiment Runtime with the Double Decomposition Approach. Theoretical Complexity Approx.: #. of unit operations in one key-switching on polynomial level, approximated using Table IV

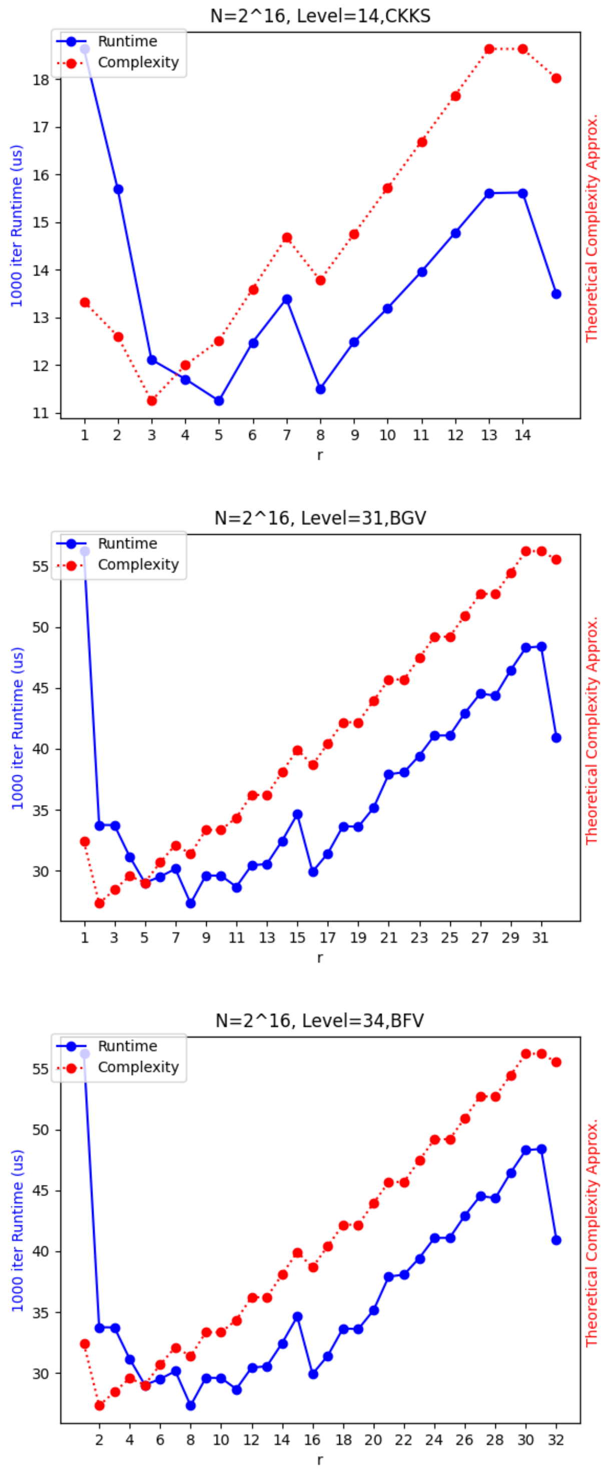


TABLE V: GPU Memory Consumption Estimation for [31]. Based on computing for a CKKS implementation with max level 32. #: number of key decompositions pre-computed

Parameters		Storage Requirement (GB)		
$\log N$	#	Offline	Online	Total
14	1	3.91	0.37	4.28
14	2	11.54	0.37	11.91
14	All	73.83	0.37	74.20
15	1	7.81	0.73	8.54
15	2	23.07	0.73	24.40
15	All	147.92	0.74	148.65
16	1	15.63	1.47	17.1
16	2	46.14	1.47	47.61
16	All	296.89	1.47	298.36

Figure 2 illustrates that approximating the runtime considering the complexity of the NTT as $O(N \log N)$ and the complexity of the Hadamard product as $O(N)$ does not accurately reflect the experimental runtime.

b) Storage Requirement: Implementing the original double decomposition method on a GPU presents significant challenges, primarily due to its substantial demand for GPU memory resources. As illustrated in Table V, the memory consumption under typical parameter settings for various schemes is considerable. Given that a single NVIDIA A100 GPU has a maximum of 80GB of memory, this limitation becomes a critical constraint. Moreover, the time-consuming nature of communication between the host and the device can become a performance bottleneck, further complicating the implementation process.

According to a previous discussion, determining the optimal width of the word decomposition r using deterministic methods is challenging. As a result, it becomes necessary to compute multiple sets of public keys **pk** and the corresponding base converters and twiddle factor tables offline. If the GPU memory is insufficient to store all precomputed resources, it necessitates the development of a highly efficient communication channel between the GPU and the host. This requirement extends beyond the scope of FHE implementation itself, demanding innovative solutions to efficiently manage data transfer and resource allocation.

V. Implementation

A. Implementation Design

In the implementation, we have systematically separated all components that can be precalculated and stored in advance, such as the public keys (**pk**) and the *BaseConverters*. This precalculation strategy allows us to optimize the computational process by minimizing the need for real-time calculations, thereby enhancing efficiency. Elements not associated with the polynomial, such as q_i^*, $\tilde{q}_i$, and $\mathbf{Q_{ibase}}/q_i$, as well as the forward and backward NTT twiddle factor tables with bases $[m_0, m_1]$ and $\mathbf{T}$, can be precomputed. These components are inherently dependent on the polynomial's characteristics and thus require real-time computation.

B. Optimizing the GPU Performance

As discussed in Section II-E, the number of kernel initializations significantly impacts runtime, making it crucial to

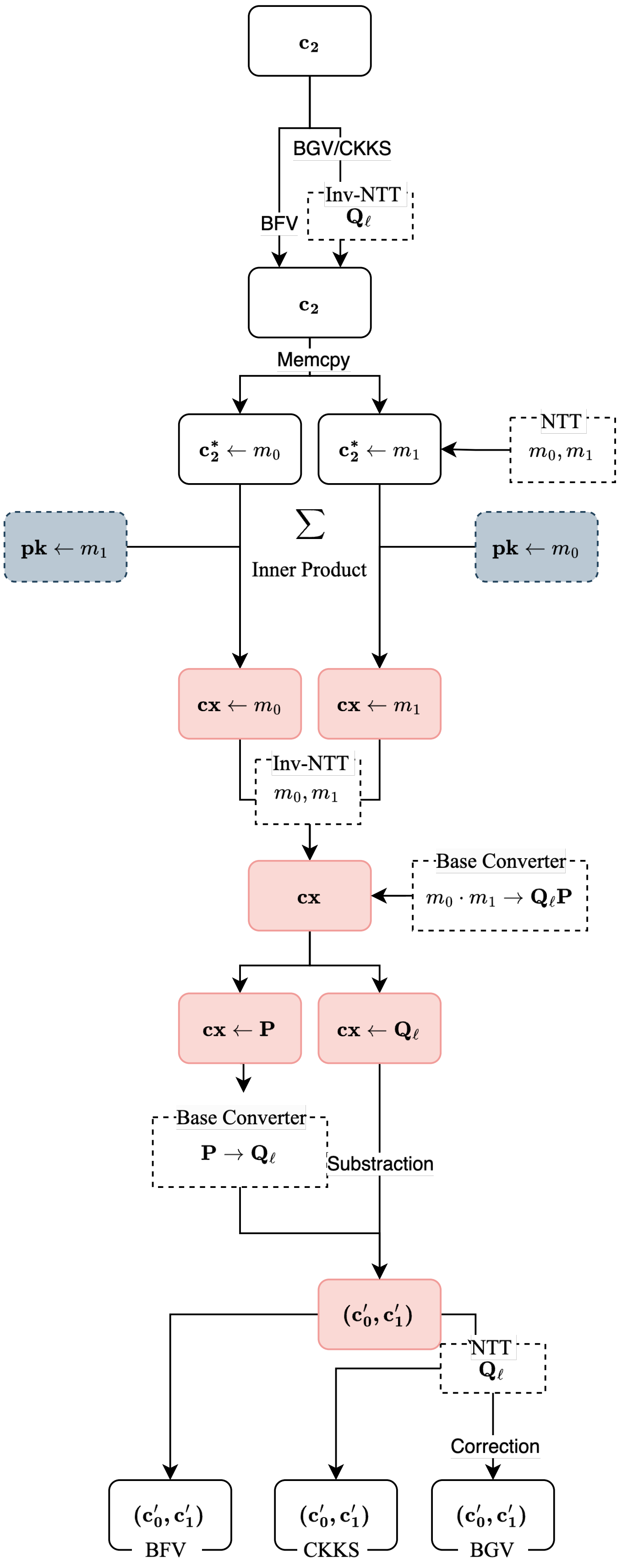

$\mathbf{c_2}$
BGV/CKKS
BFV
Inv-NTT $\mathbf{Q}_\ell$
$\mathbf{c_2}$
Memcpy
$\mathbf{c_2^*} \leftarrow m_0$
$\mathbf{c_2^*} \leftarrow m_1$
NTT m_0, m_1
$\mathbf{pk} \leftarrow m_1$
$\sum$ Inner Product
$\mathbf{pk} \leftarrow m_0$
$\mathbf{cx} \leftarrow m_0$
$\mathbf{cx} \leftarrow m_1$
Inv-NTT m_0, m_1
$\mathbf{cx}$
Base Converter $m_0 \cdot m_1 \rightarrow \mathbf{Q}_\ell\mathbf{P}$
$\mathbf{cx} \leftarrow \mathbf{P}$
$\mathbf{cx} \leftarrow \mathbf{Q}_\ell$
Base Converter $\mathbf{P} \rightarrow \mathbf{Q}_\ell$
Substraction
$(\mathbf{c}_0', \mathbf{c}_1')$
NTT $\mathbf{Q}_\ell$
Correction
$(\mathbf{c}_0', \mathbf{c}_1')$ BFV
$(\mathbf{c}_0', \mathbf{c}_1')$ CKKS
$(\mathbf{c}_0', \mathbf{c}_1')$ BGV

Fig. 3: Unified Implementation of Linear Key-Switching Method for BFV, BGV, and CKKS Schemes

TABLE VI: List of parameters used in the experiment

$\log N$	scheme	$\ell_{\max}$	$\lvert\mathbf{Q}\rvert \times$ size (bits)	$\lvert\mathbf{P}\rvert \times$ size (bits)
14	BGV	32	32×40	1×55
	CKKS	32	$1 \times 55, 31 \times 40$	1×55
	BFV	8	7×40	1×55
15	BGV	32	32×40	1×55
	CKKS	32	$1 \times 55, 31 \times 40$	1×55
	BFV	15	14×40	1×55
16	BGV	32	32×40	1×55
	CKKS	32	$1 \times 55, 31 \times 40$	1×55
	BFV	34	33×40	1×55

minimize these initializations. In linear key-switching, where many parameters are pre-determined, there is potential to compress pre-processing tasks and perform kernel fusing. This optimization is particularly beneficial because it reduces the overhead associated with multiple kernel launches, which can be a major performance bottleneck in GPU implementations.

For instance, consider a module in Algorithm 5 that invokes the NTT kernels ℓ times. By predetermining values such as $[m_0, m_1]$, we can optimize the process by introducing a fused kernel and moving this kernel call outside the loop, thus reducing the number of kernel initializations to just one.

Such optimizations are crucial in maximizing the performance of GPU-based cryptographic operations, as they leverage the parallel processing capabilities of GPUs while minimizing the latency associated with kernel management. By focusing on pre-determined parameters and efficient kernel management, we can achieve significant improvements in computational efficiency.

VI. Experiment

A. Experiment Setup

The project was implemented and tested on a single NVIDIA A100 GPU to assess the performance of the implemented schemes. Comprehensive performance benchmarks were conducted to evaluate the efficiency and scalability of these schemes. The parameters used for thorough tests across different schemes are detailed in Table VI, outlining the specific configurations used in the evaluation process.

B. Evaluation

Speed: Table VIII offers a comprehensive comparison of the runtime performance among baseline, the linear-keyswitching approach and the double decomposition method.

- **Speed Performance Improvement:** Our approach consistently demonstrates significant speed improvements over both the best results from [31] and the baseline implementation. For instance, in the BGV scheme at $\log N = 14$ of level $\ell = 32$, our method achieves a speed of 8.07 ms, representing a 1.5x improvement over [31] and a 1.6x improvement over the baseline.
- **Efficiency Across Schemes:** The table highlights that our approach is effective across different schemes. For the CKKS scheme at $\log N = 14$, our method shows a 2.0x speedup compared to the baseline for $\ell = 32$. This indicates robust performance enhancements regardless of the encryption scheme used.
- **Scalability with Parameters:** As the parameter $\log N$ increases, our approach maintains its efficiency. For example, at $\log N = 16$ in the BGV scheme, our method achieves a speed of 33.12 ms for $\ell = 32$, showing a 1.5x improvement over the baseline.
- **Consistent Gains:** Across all configurations, our method consistently outperforms the baseline and even the best setting of [31], demonstrating its reliability and effectiveness in optimizing key-switching operations. For the BFV scheme at $\log N = 14$, the speed improvement reaches up to 2.0x compared to the best decomposition strategy of [31]; For CKKS with $\log N = 14$, our speed-up reaches as high as 2.0x on level $\ell = 32$.

Speed Performances by Level: For leveled schemes such as BGV and CKKS, we examine the speed performance as the level increases. Figure 4 illustrates that our approach achieves significant runtime performance improvements, particularly for $N = 2^{14}$ and $N = 2^{15}$. Although for $N = 2^{16}$ the advantage of our method diminishes slightly, it remains closely aligned with the optimal parameter choice of r, and continues to perform well as long as the level does not grow excessively large. Specifically, our scheme demonstrates exceptionally stable and fast performance in smaller parameter sets, such as $N = 2^{14}$, where the benefits of the double decomposition approach are less evident. This stability underscores the efficiency of our method.

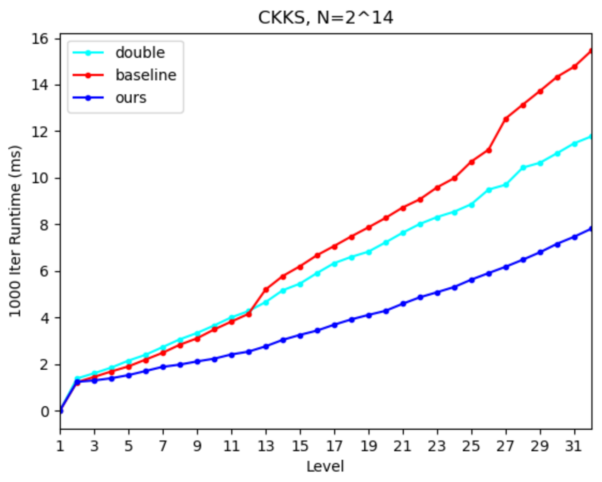
Operation Proportion Analysis: Figure 5 provides a breakdown of time consumption during the key-switching process. For the analysis of [31], the decompositions with $r = 11$ and $r = 7$ are selected because they provide the best and second-best performance, respectively, in terms of overall runtime efficiency. It is evident that both the linear and double decomposition approaches significantly reduce the time required for the $\mathbf{c_2}$ modulus switch compared to previous methods. A notable difference, however, is that the inner product time consumption for linear key-switching can be up to twice that of the original algorithm. This observation aligns with the theoretical complexity analysis.

Space Consumption: Since our scheme introduces only an additional RNS base $[m_0, m_1]$, it is sufficient to precalculate the public key **pk** to be approximately twice its original size. This is in stark contrast to the increase required if one were to pre-calculate all decomposition keys using the double decomposition method. As shown in Table VII, our approach can save around 95% of the space compared to the worst-case of [31].

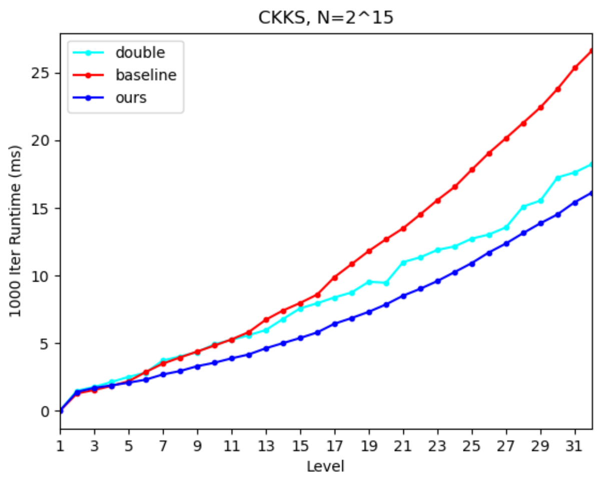
This strategy allows a single Tesla A100 GPU to efficiently store all pre-computed keys and tables, even as the computational level reaches 35 or higher. In contrast, during our experiments with the double decomposition approach, we found that the 80 GB memory was insufficient for our requirements. For $\ell > 20$, we were forced to conduct tests layer-by-layer due to GPU memory constraints. This clearly demonstrates the superior space efficiency of our method.

C. Discussion

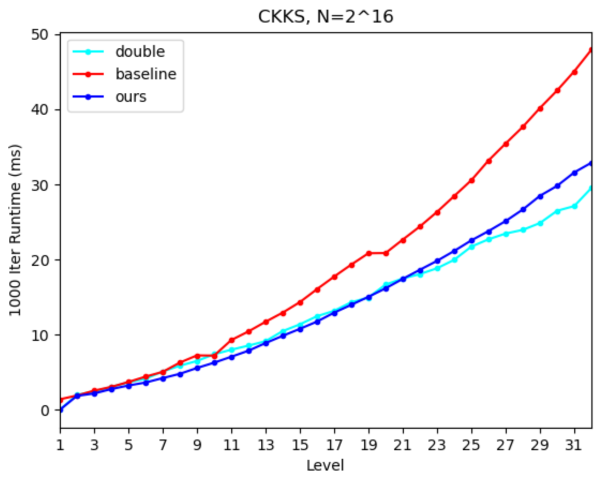
In previous sections, we demonstrated that our method significantly outperforms than the baseline approach in terms



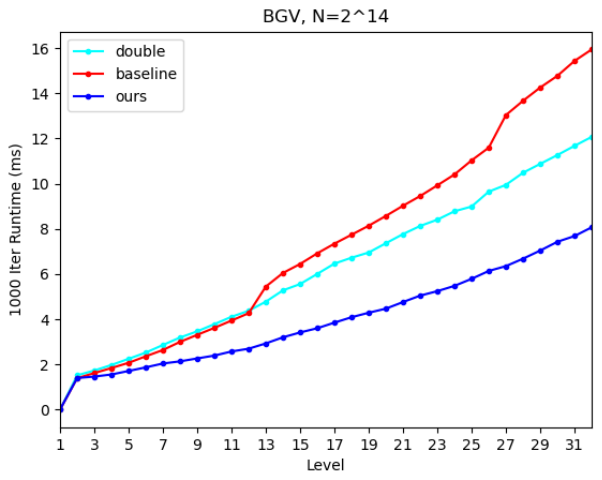
(a)



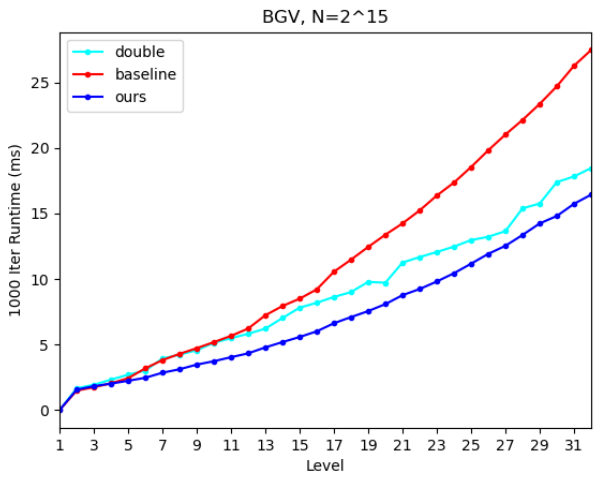
(b)



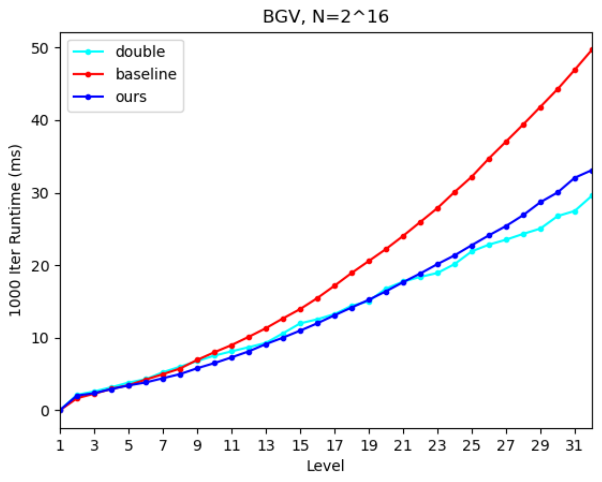
(c)



(d)



(e)



(f)

Fig. 4: Performance comparison of ours, double decomposition[31], and baseline implementations for the levelled schemes

TABLE VII: Space Consumption

$\log N$	scheme	level	single level memory usage (GB)		
			[31]	ours	space saving
14	BGV	32	11.38	0.55	95.15%
	CKKS	32	11.87	0.55	95.34%
	BFV	8	0.30	0.04	85.80%
15	BGV	32	22.76	1.10	95.15%
	CKKS	32	23.85	1.10	95.37%
	BFV	15	2.93	0.26	91.01%
16	BGV	32	45.64	2.21	95.16%
	CKKS	32	47.70	2.21	95.37%
	BFV	34	53.77	2.48	95.38%

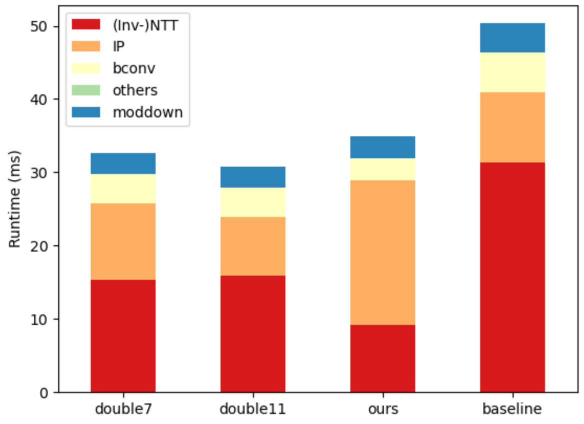


Fig. 5: Breakdown of 1000 iter runtime for CKKS level 32, $N = 2^{16}$. double7:[31] with $r = 7$; double11:[31] with $r = 11$. IP: Inner Product; bconv: Base Conversion

of speed ant NTT complexity. Compared with [31], although the double decomposition approach theoretically offers better complexity in terms of NTT, this advantage does not necessarily translate to its GPU implementations. Several challenges contribute to this discrepancy:

1) Parameter Selection: Achieving the best theoretical complexity is often impractical because determining the optimal parameters is challenging. The complexity of the parameter space makes it difficult to consistently select values that yield optimal performance
2) Storage and Memory Constraints: Even when optimal parameters are identified, they impose stringent requirements on storage and memory management. The increased demand for memory can exceed the capacity of available hardware, such as GPUs, thereby limiting the practical applicability of the approach
3) Algorithmic Complexity: The inherent complexity of the double decomposition algorithm poses significant challenges for developers in terms of GPU acceleration. The intricate nature of the algorithm complicates the implementation process, making it difficult to fully leverage the parallel processing capabilities of GPUs.

In contrast, we achieved better performance in terms of both speed and space efficiency. Our approach has the following advantages:

1) **Speed:** Achieved through GPU acceleration and optimization, our method significantly reduces computational time compared to the double decomposition approach.
2) **Space Efficiency:** Our approach is more accommodating to the memory limitations of modern single GPU cards. By optimizing memory usage and reducing the size of pre-computed keys, we ensure that our implementation fits within the constraints of available GPU memory.

Another noteworthy issue is the development burden associated with implementing the double decomposition approach. In our experience, the implementation of the method described in [31] is significantly more time-consuming. Unlike CPU development, CUDA coding and testing can be more challenging due to the GPU's less transparent nature to developers. Implementing the double decomposition requires designing additional GPU kernels, which extends the development cycle and complicates GPU optimization efforts. Moreover, kernel fusing becomes difficult due to the need to maintain the flexibility of the original algorithm. The presence of numerous undetermined parameters adds to the testing burden, as FHE schemes already involve a multitude of parameters that need to be tested. This complexity makes it more challenging to find an optimal combination of parameters that ensures efficient performance.

Our approach effectively balances simplicity and generality, taking into account the constraints inherent in GPU implementations. In real-world applications such as PPML and cloud computing, obtaining detailed hardware information from users and tuning parameters for the second decomposition can be challenging, raising additional privacy concerns. On the other hand, to maintain flexibility, one might need to precompute additional parameters, which can lead to increased computational overhead and resource consumption.

Our method is optimized for most scenarios, allowing for potential GPU acceleration optimizations without excessive precomputation. For scenarios where hardware optimization and ease of integration are crucial, the linear approach may offer more advantages due to its straightforward nature and lower resource demands. Conversely, applications requiring extensive flexibility and adaptability might benefit from a more generalized approach such as [31], though this comes with the trade-off of increased complexity and resource requirements

For $\alpha > 1$: Experimental results indicate that both the linear-key switching method and the approach described in [31] exhibit performance degradation when the parameter α exceeds 1. However, our method demonstrates relative advantages, as it handles smaller parameters more effectively. In subsequent research, we aim to tackle this challenge by integrating other techniques that have shown promise in similar contexts. Through these efforts, we hope to develop more robust solutions that maintain high performance even when α exceeds 1.

VII. Conclusion

In conclusion, we have introduced a novel algorithm that achieves linear NTT complexity for key-switching, demon-

TABLE VIII: Performance comparison of the implemented linear and double-decomposition approaches

			Best of [31]			Baseline	Ours		
$\log N$	scheme	ℓ	r	$\lvert\mathbf{T}\rvert$	time (ms)	time (ms)	time (ms)	speed (vs. [31])	speed (vs. baseline)
14	BGV	32	11	9	12.12	13.19	8.07	x1.5	x1.6
		24	13	10	8.68	9.18	5.48	x1.6	x1.7
		16	9	8	6.16	6.91	3.60	x1.7	x1.9
		8	9	8	3.19	2.98	2.14	x1.5	x1.4
	CKKS	32	11	9	11.78	15.45	7.82	x1.5	x2.0
		24	13	10	8.55	10.00	5.32	x1.6	x1.9
		16	9	8	6.05	6.68	3.45	x1.8	x1.9
		8	9	8	3.13	2.84	1.99	x1.6	x1.4
	BFV	8	5	5	2.86	2.33	1.45	x2.0	x1.6
15	BGV	32	5	5	18.51	27.41	16.43	x1.1	x1.7
		24	7	6	12.46	17.34	10.44	x1.2	x1.7
		16	7	6	8.19	9.25	6.01	x1.4	x1.5
		8	5	5	4.23	4.31	3.12	x1.4	x1.4
	CKKS	32	5	5	18.24	26.65	16.13	x1.1	x1.7
		24	7	6	12.16	16.57	10.26	x1.2	x1.6
		16	7	6	8.00	8.61	5.80	x1.4	x1.5
		8	5	5	4.02	3.95	2.95	x1.4	x1.3
	BFV	15	4	4	6.91	7.65	4.36	x1.6	x1.8
16	BGV	32	11	9	34.62	49.62	33.12	x1.0	x1.5
		24	5	5	20.17	30.04	21.36	x0.9	x1.4
		16	6	6	12.55	15.26	11.99	x1.0	x1.3
		8	5	5	6.00	5.68	4.99	x1.2	x1.1
	CKKS	32	11	9	29.53	47.94	32.89	x0.9	x1.5
		24	13	10	20.96	28.53	21.15	x1.0	x1.3
		16	6	6	13.81	14.34	11.75	x1.2	x1.2
		8	5	5	5.88	5.06	4.80	x1.2	x1.1
	BFV	34	5	5	36.07	54.62	32.96	x1.1	x1.7

strating significant potential for GPU and hardware optimization. This advancement is particularly relevant in the context of cryptographic operations, where efficiency and resource management are critical. Our approach not only simplifies the computational process but also enhances the feasibility of implementing these operations on GPUs, thereby optimizing hardware utilization.

Furthermore, we have completed a GPU implementation of the methodology outlined in [31]. This implementation serves as a practical demonstration of the algorithm's capabilities, allowing others to explore and build upon our work.

Our comprehensive experiments and comparisons underscore the effectiveness of our approach. The results indicate that the simplicity inherent in our method not only achieves comparable improvements to existing techniques, but often surpasses them. This is achieved through enhanced preprocessing, which significantly reduces the computational overhead and conserves GPU memory resources. By streamlining the preprocessing phase, our algorithm minimizes the memory footprint, thereby addressing one of the key challenges in GPU-based implementations. In general, the proposed algorithm represents a significant contribution to the field, offering a balance between simplicity and performance that is crucial to advance cryptographic computations on modern hardware platforms.

Acknowledgement

This research is supported in part by the Jiangsu Province 100 Foreign Experts Introduction Plan BX2022012. This research is also supported in part by TÜBİTAK Projects 2232-118C332 and 1001-121F348.

References

[1] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, et al. Openfhe: Open-source fully homomorphic encryption library. In *proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography*, pages 53–63, 2022.

[2] Ahmad Al Badawi, Bharadwaj Veeravalli, Jie Lin, Nan Xiao, Matsumura Kazuaki, and Aung Khin Mi Mi. Multi-gpu design and performance evaluation of homomorphic encryption on gpu clusters. *IEEE Transactions on Parallel and Distributed Systems*, 32(2):379–391, 2020.

[3] Ahmad Al Badawi, Bharadwaj Veeravalli, Jie Lin, Nan Xiao, Matsumura Kazuaki, and Aung Khin Mi Mi. Multi-gpu design and performance evaluation of homomorphic encryption on gpu clusters. *IEEE Transactions on Parallel and Distributed Systems*, 32(2):379–391, 2020.

[4] Ahmad Al Badawi, Bharadwaj Veeravalli, Chan Fook Mun, and Khin Mi Mi Aung. High-performance fv somewhat homomorphic encryption on gpus: An implementation using cuda. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 70–95, 2018.

[5] Pedro Geraldo MR Alves, Jheyne N Ortiz, and Diego F Aranha. Faster homomorphic encryption over gpgpus via hierarchical dgt. In *International Conference on Financial Cryptography and Data Security*, pages 520–540. Springer, 2021.

[6] Wei Ao and Vishnu Naresh Boddeti. AutoFHE: Automated adaption of CNNs for efficient evaluation over FHE. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2173–2190, Philadelphia, PA, August

2024. USENIX Association.

[7] Y. Aono, T. Hayashi, L. Wang, and S. Moriai. Privacy-preserving deep learning via additively homomorphic encryption. ieee. *transactions on information forensics and*, 13(5):1333–1345.

[8] L.J. Aslett, P.M. Esperança, and C.C. Holmes. Encrypted statistical machine learning: new privacy preserving methods. arXiv preprint arXiv:1508.06845.

[9] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[10] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on computing*, 43(2):831–871, 2014.

[12] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 1–12, 2014.

[13] H. Chabanne, A. Wargny, J. Milgram, C. Morel, and E. Prouff. Privacy-preserving classification on deep neural network. cryptology eprint archive.

[14] Hao Chen, Kim Laine, and Rachel Player. 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*, pages 3–18. Springer, 2017.

[15] Huili Chen, Rosario Cammarota, Felipe Valencia, Francesco Regazzoni, and Farinaz Koushanfar. Ahec: End-to-end compiler framework for privacy-preserving machine learning acceleration. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[16] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2018/931, 2018.

[17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[18] W. Dai and B. Sunar. cuhe: A homomorphic encryption accelerator library. *Cryptography and Information Security in the Balkans: Second International Conference, BalkanCryptSec 2015*, 2 (pp:169–186.

[19] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[20] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[21] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. Cryptology ePrint Archive, Paper 2012/099, 2012. https://eprint.iacr.org/2012/099.

[22] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92. Springer, 2013.

[23] Jia-Zheng Goey, Wai-Kong Lee, Bok-Min Goi, and Wun-She Yap. Accelerating number theoretic transform in gpu platform for fully homomorphic encryption. *The Journal of Supercomputing*, 77:1455–1474, 2021.

[24] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. Cryptology ePrint Archive, Paper 2018/117, 2018. https://eprint.iacr.org/2018/117.

[25] Shai Halevi and Victor Shoup. Design and implementation of a homomorphic-encryption library. *IBM Research (Manuscript)*, 6(12-15):8–36, 2013.

[26] Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. Cryptology ePrint Archive, Paper 2019/688, 2019. https://eprint.iacr.org/2019/688.

[27] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, Baltimore, MD, August 2018. USENIX Association.

[28] A. Khedr, G. Gulak, and V. Vaikuntanathan. Shield: scalable homomorphic implementation of encrypted data-classifiers. ieee. *Transactions on*, 65(9):2848–2858.

[29] Jongmin Kim, Wonseok Choi, and Jung Ho Ahn. Cheddar: A swift fully homomorphic encryption library for cuda gpus. *arXiv preprint arXiv:2407.13055*, 2024.

[30] Miran Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Accelerating he operations from key decomposition technique. In *Annual International Cryptology Conference*, pages 70–92. Springer, 2023.

[31] Miran Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Accelerating HE operations from key decomposition technique. Cryptology ePrint Archive, Paper 2023/413, 2023. https://eprint.iacr.org/2023/413.

[32] Johannes Mono and Tim Güneysu. A new perspective on key switching for BGV-like schemes. Cryptology ePrint Archive, Paper 2023/1642, 2023. https://eprint.iacr.org/2023/1642.

[33] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124, 2011.

[34] Wei Wang, Zhilu Chen, and Xinming Huang. Accelerating leveled fully homomorphic encryption using gpu. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2800–2803. IEEE, 2014.

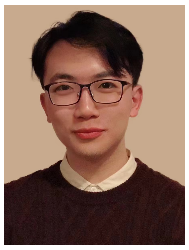
[35] Wei Wang, Yin Hu, Lianmu Chen, Xinming Huang, and

Berk Sunar. Accelerating fully homomorphic encryption using gpu. In *2012 IEEE conference on high performance extreme computing*, pages 1–5. IEEE, 2012.

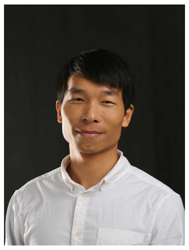
[36] Hao Yang, Shiyu Shen, Wangchen Dai, Lu Zhou, Zhe Liu, and Yunlei Zhao. Phantom: a cuda-accelerated word-wise homomorphic encryption library. *IEEE Transactions on Dependable and Secure Computing*, 2024.

Shutong Jin received her B.Eng degree from Department of Electrical Engineering, City University ofHong Kong in 2020. She is now a Ph.D. candidate supervised by Prof. Ray Cheung. Her research interests include privacy-preserving deep learning, GPU acceleration and cryptography.

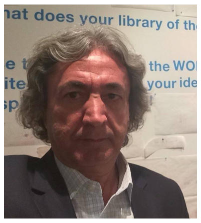
Zhen Gu is a Research Scientist in the DAMO Academy of Alibaba Group. He earned his B.S. degree in the Department of Microelectronics and Nanoelectronics from Tsinghua University, Beijing, China; and his Ph.D. degree in the School of Integrated Circuits from Tsinghua University, Beijing, China. His research interests are in privacy-enhancing technologies (PETs) and hardware acceleration of secure computation. Dr. Gu has published papers on both hardware acceleration and application protocols in conferences like DAC, NDSS and VLDB, etc.



Guangyan Li received the B.Eng degree in 2020 from the Department of Electrical Engineering, City University of Hong Kong. He is now pursuing the PhD degree in the Department of Electrical Engineering from City University of Hong Kong. His research interests include reconfigurable computing with FPGA, and post-quantum cryptography algorithm design.



Donglong Chen received the PhD degree from the Department of Electronic Engineering, City University of Hong Kong, in 2015. He was a visiting research scholar of COSIC, KU Leuven, Belgium, in 2013. After completing his PhD degree study, he spent four years with the industry including Huawei Technology Co., Ltd. and Tencent Technology Co., Ltd. He is currently an associate professor with the Faculty of Science and Technology, BNU-HKBU United International College (UIC), China. His research interests include cryptographic engineering, software/hardware co-design for AI algorithms, and privacy computing.



Çetin Kaya Koç received his Ph.D. in Electrical & Computer Engineering from the University of California Santa Barbara. His research interests include cryptographic engineering, finite field arithmetic, random number generators, homomorphic encryption and machine learning. Koç is the co-founder of the Conference on Cryptographic Hardware and Embedded Systems (1999) which is the second largest cryptography conference in the world. Koç also is the founding editor-in-chief of the Journal of Cryptographic Engineering, published by Springer since 2011, with focus on cryptographic hardware and software development. Koç was elected as an IEEE Fellow in 2007 and IEEE Life Fellow in 2023 for his contributions to cryptographic engineering.

Ray C. C. Cheung received the Ph.D. degrees in computing from Imperial College London, London, U.K., in 2007. He conducted his postdoctoral research work with UCLA and his visiting fellowship with Princeton University. He is a Professor with the Department of Electrical Engineering, and an Associate Provost (Digi- tal Learning), CityUHK. His current research interests include cryptographic processor designs and embedded system designs. He is now an Associate Editor of the Journal of Cryptographic Engineering, Springer, and Frontiers in High Performance Computing. He served as the Technical Chair of FPT'02, the General Chair/Co-Chair of ARC'12, FPT'22, and ASAP'24. He is currently the Chair of the IEEE Hong Kong Section.

Wangchen Dai received the Ph.D. degree in electronic engineering from the City University of Hong Kong in 2018. After that, he had appointments at Hardware Security Lab, Huawei Technologies Company Ltd., the Department of CSSE, Shenzhen University and Zhejiang Lab, Hangzhou, China. He is currently an associate professor at Sun Yat-Sen University. His research interests include cryptographic hardware and embedded systems, fully holomorphic encryption, and reconfigurable computing.

Appendix

This appendix provides the detailed implementation algorithm for the method detailed in Section IV, serving as a supplementary resource. Algorithms 4 through 7 present our GPU implementations for all word-type schemes in a clear, step-by-step manner. This detailed exposition aims to enhance understanding and facilitate replication of implementation on GPUs.

Algorithm 4: Double Decomposition: Key Decomposition

```
Data: $\mathbf{pk_0}[c_0|c_1], \mathbf{pk_1}[c_0|c_1], ..., \mathbf{pk_{\beta-1}}[c_0|c_1]$ on $\mathcal{R}^2_{\mathbf{Q}_\ell \mathbf{P}}$
Result: $\begin{bmatrix} v_{0,0} & \cdots & v_{0,d-1} \\ \vdots & \ddots & \vdots \\ v_{\beta-1,0} & \cdots & v_{\beta-1,d-1} \end{bmatrix}, v_{i,j} = v_{i,j}[c_0|c_1]$ on $\mathcal{R}^2_{\mathbf{T}}$
for $i \leftarrow 0$ to $\beta - 1$ do
    for $j \leftarrow 0$ to $d - 1$ do
        ibase $\leftarrow [\tilde{q}_{r \cdot j}, \tilde{q}_{r \cdot j + r}]$;
        obase $\leftarrow \mathbf{T}$ ;
        $u_{i,j}[c_0] \leftarrow$ part $\mathbf{pk}_i[c_0]$ on $[\tilde{q}_{r \cdot j}, \tilde{q}_{r \cdot j + r}]$;
        $u_{i,j}[c_1] \leftarrow$ part $\mathbf{pk}_i[c_1]$ on $[\tilde{q}_{r \cdot j}, \tilde{q}_{r \cdot j + r}]$;
        $v_{i,j}[c_0] \leftarrow$BaseConversion($u_{i,j}[c_0]$, ibase, obase);
        $v_{i,j}[c_1] \leftarrow$BaseConversion($u_{i,j}[c_1]$, ibase, obase);
        $v_{i,j}[c_0] \leftarrow$NTT($v_{i,j}[c_0], \mathbf{T}$);
        $v_{i,j}[c_1] \leftarrow$NTT($v_{i,j}[c_1], \mathbf{T}$);
    end
end
```

Algorithm 5: Double Decomposition: $\mathbf{c_2}$ mod-switch

```
Data: $\mathbf{c_2}$ on $\mathcal{R}_{\mathbf{Q}_\ell}$
Result: $\mathbf{b_0}, \mathbf{b_1}, .., \mathbf{b}_{\beta-1}$ on $\mathcal{R}_{\mathbf{T}}$
if scheme==ckks or scheme==bgv then
    $c' \leftarrow$Inv-NTT($\mathbf{c_2}$, $\mathbf{Q}_\ell$);
else
    $c' \leftarrow \mathbf{c_2}$;
end
if $\alpha = 1$ then
    for $i \leftarrow 0$ to $\ell - 1$ do
        $\mathbf{b_i} \leftarrow [[c']_{q_i}]_{\mathbf{T}}$ ;
        $\mathbf{b_i} \leftarrow$NTT($\mathbf{b_i}, \mathbf{T}$);
    end
else
    for $i \leftarrow 0$ to $\beta - 1$ do
        ibase $\leftarrow [q_{\alpha \cdot i}, q_{\alpha \cdot i + \alpha}]$;
        obase $\leftarrow \mathbf{T}$ ;
        $c'_i \leftarrow$ part $c'$ on $[q_{\alpha \cdot i}, q_{\alpha \cdot i + \alpha}]$;
        $\mathbf{b_i} \leftarrow$BaseConversion($c'_i$, ibase, obase);
        $\mathbf{b_i} \leftarrow$NTT($\mathbf{b_i}, \mathbf{T}$);
    end
end
```

Algorithm 6: Double Decomposition: Inner Product

```
Data: $\mathbf{b_0}, \mathbf{b_1}, .., \mathbf{b_{\beta-1}}$ on $\mathcal{R}_{\mathbf{T}}$;
      $\begin{bmatrix} v_{0,0} & \cdots & v_{0,d-1} \\ \vdots & \ddots & \vdots \\ v_{\beta-1,0} & \cdots & v_{\beta-1,d-1} \end{bmatrix}, v_{i,j} = v_{i,j}[c_0|c_1]$ on $\mathcal{R}^2_{\mathbf{T}}$
Result: $\mathbf{cx}[c_0|c_1]$ on $\mathcal{R}^2_{\mathbf{Q}_\ell \mathbf{P}}$
for $j \leftarrow 0$ to $d - 1$ do
    $\mathbf{cx}_j[c_0|c_1] \leftarrow 0$ on $\mathcal{R}^2_{\mathbf{T}}$;
    for $i \leftarrow 0$ to $\beta - 1$ do
        $\mathbf{cx}_j[c_0] \leftarrow \mathbf{cx}_j[c_0] + \mathbf{b}_i \cdot v_{i,j}[c_0]$ ;
        $\mathbf{cx}_j[c_1] \leftarrow \mathbf{cx}_j[c_0] + \mathbf{b}_i \cdot v_{i,j}[c_1]$;
    end
    $\mathbf{cx}_j[c_0] \leftarrow$ Inv-NTT($\boldsymbol{cx}_j[c_0], \mathbf{T}$) ;
    $\mathbf{cx}_j[c_1] \leftarrow$ Inv-NTT($\boldsymbol{cx}_j[c_1], \mathbf{T}$) ;
    ibase $\leftarrow \mathbf{T}$;
    obase $\leftarrow [\tilde{q}_{r \cdot j}, \tilde{q}_{r \cdot j + r}]$;
    part $\mathbf{cx}[c_0]$ on $[\tilde{q}_{r \cdot j}, \tilde{q}_{r \cdot j + r}] \leftarrow$
        BaseConversion($\boldsymbol{cx}_j[c_0]$, ibase, obase);
    part $\mathbf{cx}[c_1]$ on $[\tilde{q}_{r \cdot j}, \tilde{q}_{r \cdot j + r}] \leftarrow$
        BaseConversion($\boldsymbol{cx}_j[c_1]$, ibase, obase);
end
```

Algorithm 7: Double Decomposition: Mod-down

```
Data: $\mathbf{cx}[c_0|c_1]$ on $\mathcal{R}^2_{\mathbf{Q}_\ell \mathbf{P}}$
Result: $\mathbf{c}'_2[c_0|c_1]$ on $\mathcal{R}^2_{\mathbf{Q}_\ell}$
$\Delta[c_0] \leftarrow [\mathbf{cx}[c_0]]_{\mathbf{P}}$;
$\Delta[c_1] \leftarrow [\mathbf{cx}[c_1]]_{\mathbf{P}}$;
if $\alpha = 1$ then
    for $i \leftarrow 0$ to $\ell - 1$ do
        $\mathbf{b_i} \leftarrow [[c']_{q_i}]_{\mathbf{T}}$ ;
        $\Delta[c_0] \leftarrow [\Delta[c_0]]_{\mathbf{Q}_\ell}$ ;
        $\Delta[c_1] \leftarrow [\Delta[c_1]]_{\mathbf{Q}_\ell}$ ;
    end
else
    for $i \leftarrow 0$ to $\beta - 1$ do
        ibase $\leftarrow \mathbf{P}$;
        obase $\leftarrow \mathbf{Q}_\ell$ ;
        $\Delta[c_0] \leftarrow$BaseConversion($\Delta[c_0]$, ibase, obase) ;
        $\Delta[c_1] \leftarrow$BaseConversion($\Delta[c_1]$, ibase, obase) ;
    end
end
if scheme==ckks then
    $\mathbf{c}'_2[c_0|c_1] \leftarrow$
        ModDownKernel($\boldsymbol{cx}[c_0|c_1], \Delta[c_0|c_1]$) ;
    $\mathbf{c}'_2[c_0|c_1] \leftarrow$ NTT($\boldsymbol{c}'_2[c_0|c_1], \mathbf{Q}_\ell$) ;
else if scheme==bgv then
    $\mathbf{c}'_2[c_0|c_1] \leftarrow$
        BGVModDownKernel($\boldsymbol{cx}[c_0|c_1], \Delta[c_0|c_1]$) ;
    $\mathbf{c}'_2[c_0|c_1] \leftarrow$ NTT($\boldsymbol{c}'_2[c_0|c_1], \mathbf{Q}_\ell$) ;
else if scheme==bfv then
    $\mathbf{c}'_2[c_0|c_1] \leftarrow$
        ModDownKernel($\boldsymbol{cx}[c_0|c_1], \Delta[c_0|c_1]$) ;
```