Revisiting Key Switching Techniques with Applications to Light-Key FHE

Ruida Wang[1,2], Zhihao Li[1,2], Benqiang Wei[1,2], Chunling Chen[1,2], Xianhui Lu[1,2*], and Kunpeng Wang[1,2*]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
luxianhui@iie.ac.cn; wangkunpeng@iie.ac.cn

Abstract. Fully Homomorphic Encryption (FHE) allows for data processing while it remains encrypted, enabling privacy-preserving outsourced computation. However, FHE faces challenges in real-world applications, such as communication overhead and storage limitations, due to the large size of its evaluation key.
This paper revisits existing *key switching* algorithms widely used in FHE, which may account for over 90% of the total evaluation key size. Although these algorithms work towards the same goal, they differ significantly in functionality, computational complexity, noise management and key size. We close their functional gap and reanalyze them under a common standard, proposing theorems and comparative results to provide a flexible time-space trade-off when designing FHE applications.
To validate the efficacy of our theoretical results, we propose a light-key bootstrapping method using a lower-sized key switching variant. This approach reduces the key size of the well-known GINX bootstrapping by a factor of 88.8%. It also outperforms the state-of-the-art light-key FHE by reducing 48.4% bootstrapping key size and 8% transfer key size.

Keywords: FHE · Key Switching · Light-Key Bootstrapping.

1 Introduction

Fully Homomorphic Encryption (FHE) allows data to be processed while encrypted, enabling users to delegate computation to an untrusted party without the risk of data leakage. This opens up the potential for privacy-preserving outsourced computation in various applications, such as cloud computing [1,19], the internet of things (IoT) [26,29] and machine learning [20,10]. The process involves the client (data owner) encrypting their sensitive data, generating the necessary evaluation keys for homomorphic operations, and transmitting them to the server (computing party). The server performs homomorphic evaluations on the ciphertext and returns the encrypted results, as shown in fig.1.

One issue faced by Fully Homomorphic Encryption (FHE) is the storage and the communication cost. FHE is based on lattice encryption schemes, resulting in

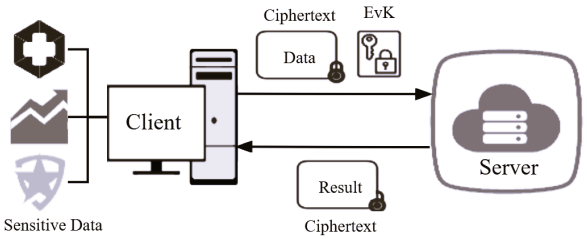

Fig. 1. Client-server model of FHE applications. Evk denotes the evaluation keys.

large ciphertext and key sizes. In word-wise encryption schemes, the evaluation keys often have sizes of gigabytes [14,4,16,17]. While FHEW-like bit-wise encryption schemes reduce the evaluation key size by one order of magnitude, they still face limitations in real-world applications due to their key size of about 200 MB. More precisely, there is a strong preference for clients to generate and transmit keys with the smallest possible size. This is due to the fact that clients typically operate on devices with constrained computing power and limited storage space, sometimes even on mobile devices [13,28].

From the server's perspective, research has demonstrated that hardware acceleration can yield over a ten times boost in the efficiency of homomorphic encryption operations [15,30,25]. However, these solutions are memory-constrained due to their limited on-chip storage. These challenges promote us to explore techniques to reduce the size of evaluation keys.

This paper concentrates on the key switching algorithm, whose key size may account for over 90 % of the total evaluation key in FHEW-like schemes, as shown in tab.1.

Methods	Evaluation key size	Key switching key size	Transfer key size
GINX ([21])	250 MB	229.1 MB (91.6%)	16.48 MB
LFHE ([18])	175 MB	84.6 MB (48.3%)	881 KB
GINX_{our}	27.91 MB	27.2 KB (0.1 %)	13.96 MB
LFHE_{our}	90.38 MB	54.2 KB (0.06 %)	810.1 KB

Table 1. The proportion of key switching key size in the total evaluation key size of different bootstrapping methods. The parameters resources is within brackets. In the transfer model [18], the transfer key is a seed of the evaluation key. Sec.6.4 provides a detailed description of the transfer model.

Key switching is an essential operation in FHEW-like cryptosystems that enables changing the encryption key without revealing the plaintext. Various types of key switching have been described in this literature, including LWE-to-(R)LWE key switching, and RLWE-to-RLWE key switching. Chillotti et al.

shows that the former scheme can evaluate a linear Lipschitz morphism on ciphertext almost for free during switching keys [7]. Depending on the confidentiality of the morphism, it can be further divided into public functional key switching and private functional key switching. Even for the same switching type, there are different computation methods, these algorithms differ in functionality, key size, computational complexity, and noise management. A unified comparison is currently lacking, and there is no theoretical basis for selecting proper key switching algorithms when designing FHE applications. This motivates us to comprehensively revisit known key switching algorithms.

Functional Key Switching Algorithms. Our first contribution is to fill the functional gap in key switching algorithms. TFHE's key switching can compute a linear Lipschitz morphism while switching keys [7]. This property is not presented in the LWE-to-LWE key switching proposed by Chen et al. [3], or the commonly used RLWE-to-RLWE key switching algorithm. We fill this gap by decomposing all key switching algorithms into *gadget products*[3], and embedding the linear Lipschitz morphism in it. The linear property ensures that the morphism can be correctly calculated by scalar multiplication, while the Lipschitz property helps manage noise growth. As a result, we provide functional variants of all known key switching algorithms, which may have independent interests beyond this paper. For instance, we demonstrate that the scheme switching algorithm [9] (or the same EvalSquareMult algorithm [18]) can be regarded as a specific case of our proposed RLWE-to-RLWE private functional key switching algorithm for the morphism $f(x) = \mathbf{sk} \cdot x$, where $\mathbf{sk}$ is the secret key.

Comparison Between Key Switching Algorithms. Comparing key switching algorithms can be challenging since they are proposed and analyzed using different baselines, such as algebraic structures, the key distributions, and the gadget decomposition methods[4]. In this work, we present a comprehensive reanalysis of the existing key switching algorithms and our proposed functional variants under a common standard. We use the power of two cyclotomic ring, which is commonly used in FHE schemes, binary key distribution, and the canonical approximate gadget decomposition [7]. We propose noise growth formulas and provide performance data in terms of key sizes and computational complexity. Our work serves as a theoretical basis for the practical selection of key switching algorithms when designing FHE applications.

Light-Key Bootstrapping Algorithm. To validate the efficacy of our theoretical results, we propose the light-key bootstrapping variants using a lower-sized key switching algorithm. For the well-known GINX bootstrapping, this approach reduces the bootstrapping key size by 88.8 % and the transfer key

[3] The gadget product is the computational units for scalar multiplication in FHEW-like cryptosystems

[4] The gadget decomposition is a technique used to decompose large numbers into smaller digits. This helps control error growth in FHE algorithms.

size by 15.3 %. For the state-of-the-art light-key bootstrapping, this approach outperforms Kim et al.'s LFHE method [18] by reducing 48.4 % bootstrapping key size and 8 % transfer key size.

Related Work. Fig.1 illustrates that the client must generate and transmit two components: the ciphertext and the evaluation keys. This paper focuses on reducing the size of the evaluation key. However, the ciphertext size is also significantly larger than plaintext due to the lattice-based encryption. Currently, Naehrig et al. have introduced techniques named hybrid homomorphic encryption (HHE or transciphering) [24,2,11,8]. This technique allows the client to encrypt messages with a symmetric cipher. The server then evaluates the decryption circuit homomorphically to obtain the ciphertext under HE form for further processing. Our future work involves integrating HHE with our research, to develop fully homomorphic encryption applications with minimal transmission size.

Organization. The rest of the paper is organized as follows: sec.2 reviews the notations and crypto primitives; sec.3 revisits the gadget product as the basic computational unit of key switching algorithms; sec.4 and sec.5 analyzes the LWE-to-LWE key switching algorithms and RLWE-to-RLWE key switching algorithms, respectively; sec.6 constructs the light-key bootstrapping algorithm based on the analysis results; sec.7 concludes the paper.

2 Preliminaries

2.1 Notations

Let $\mathbb{A}$ be a set. Define $\mathbb{A}^n$ as the set of vectors with n elements in $\mathbb{A}$, $\mathbb{A}_q$ as the set $\mathbb{A}$ module q, where the elements' scope is $[-q/2, q/2) \cap \mathbb{A}$. Use $\mathbb{Z}$ to denote the set of integers, $\mathbb{R}$ to denote the set of real numbers, and $\mathbb{B} = \mathbb{Z}_2$ represents the set of binary numbers. Denote $\mathcal{R}$ as the set of integer coefficient polynomials modulo $X^N + 1$, where N is a power of 2 Then $\mathcal{R}$ is the $2N$-th cyclotomic ring.

Use regular letters to represent (modular) integers like $a \in \mathbb{Z}_q$, while bold letters to represent polynomials $\mathbf{a} \in \mathcal{R}$ or vectors $\mathbf{a} \in \mathbb{Z}^n$. The notation a_i refers to the i-th coefficient/term of $\mathbf{a}$. The floor, ceiling, and rounding functions are written as $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$ $\lfloor \cdot \rceil$, respectively. A function f is R-Lipschitz means that it satisfies $\|f(x) - f(y)\|_\infty \leq R\|x - y\|_\infty$, where $\| \cdot \|_\infty$ is the infinity norm.

2.2 Gadget Decomposition

Given a gadget vector $\mathbf{v} = (v_0, v_1, ..., v_{l-1})$, the gadget decomposition of a ring element $\mathbf{t} \in R$ is to find $(\mathbf{t}_0, ..., \mathbf{t}_{l-1})$ to minimize the decomposition error $\varepsilon_{\mathsf{gadget}}(\mathbf{t}) = \sum_i v_i \mathbf{t}_i - \mathbf{t}$. ϵ denotes its infinite norm, that is, $\| \sum_i v_i \mathbf{t}_i - \mathbf{t} \|_\infty \leq \epsilon$. In this paper, we use the canonical approximate gadget decomposition, where $\mathbf{v} = (\lceil \frac{q}{B^l} \rceil, \lceil \frac{q}{B^l} \rceil B, ..., \lceil \frac{q}{B^l} \rceil B^{l-1})$, thus $\epsilon \leq \frac{1}{2} \lceil \frac{q}{B^l} \rceil$. We say B is the gadget base and l is the gadget length.

2.3 Learning with Errors

The security of FHEW-like cryptosystem is based on the (ring) learning with errors problem [27,22]. We summarize the three kinds of ciphertexts as follow:

- **LWE:** Giving positive integers n and q, the LWE encryption of the message $m \in \mathbb{Z}$ is a vector $(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, where $b = -\mathbf{a} \cdot \mathbf{sk} + m + e$. The vector $\mathbf{a}$ is uniformly sampled from $\mathbb{Z}_q^n$, the secret key $\mathbf{sk}$ is sampled from a key distribution χ, the error e is sampled from an error distribution χ'.
- **RLWE:** RLWE is a ring version of LWE on $\mathcal{R}_q$. The RLWE encryption of the message $\mathbf{m} \in \mathcal{R}_q$ is a pair $(\mathbf{a}, \mathbf{b}) \in \mathcal{R}_q^{n+1}$, where $\mathbf{b} = -\mathbf{a} \cdot \mathbf{sk} + \mathbf{m} + \mathbf{e}$. The vector $\mathbf{a}$ is uniformly sampled from $\mathcal{R}_q$, the secret key $\mathbf{sk}$ is sampled from a key distribution χ, and each coefficient of the error e_i is sampled from χ'.
- **RGSW:** The RGSW encryption of the message $\mathbf{m} \in \mathcal{R}_q$ can be expressed as: $\mathrm{RGSW}_{\mathbf{sk}}(\mathbf{m}) = (\mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{sk} \cdot \mathbf{m}), \mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{m}))$, where RLWE′ is the gadget RLWE ciphertext defined as follows:

Given a gadget vector $\mathbf{v} = (v_0, v_1, ..., v_{l-1})$, the notion (R)LWE′ refers to the gadget (R)LWE ciphertext is defined as:

$$\mathrm{LWE}'_{\mathbf{sk}}(m) = (\mathrm{LWE}_{\mathbf{sk}}(v_0 \cdot m), \mathrm{LWE}_{\mathbf{sk}}(v_1 \cdot m), ..., \mathrm{LWE}_{\mathbf{sk}}(v_{l-1} \cdot m)),$$

$$\mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{m}) = (\mathrm{RLWE}_{\mathbf{sk}}(v_0 \cdot \mathbf{m}), \mathrm{RLWE}_{\mathbf{sk}}(v_1 \cdot \mathbf{m}), ..., \mathrm{RLWE}_{\mathbf{sk}}(v_{l-1} \cdot \mathbf{m})).$$

Remark 1. These definitions (following Micciancio and Polyakov [23]) use different notions compared to the original TFHE papers [5,6,7]. Specifically, TFHE uses real torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ and $\mathbb{T}_N[X] = \mathcal{R}/\mathbb{Z}$ to describe the message and ciphertext spaces, but implements $\mathbb{T}$ by $\mathbb{Z}_q$ with $q = 2^{32}$ or $q = 2^{64}$. Thus we straightforwardly use $\mathbb{Z}_q$ instead of $\mathbb{T}$.

Remark 2. In FHEW-like cryptosystem, the gadget (R)LWE is mainly used as the evaluation key and appears as an auxiliary input in algorithms such as key switching. To simplify the presentation and facilitate the understanding of the key switching algorithm, which is the main focus of this paper, we provide a formal definition and notation of gadget (R)LWE.

2.4 Bootstrapping

The error rate of the LWE/RLWE ciphertext will significantly affect the decryption failure probability, which can be calculated by $1 - \mathrm{erf}\left(\frac{q}{8\sqrt{2}\sigma}\right)$, where σ is the standard deviation of the error. We then introduce the bootstrapping algorithm to reduce the error rate. FHEW-like bootstrapping can evaluate a 1-in/1-out LUT function while refreshing ciphertext noise. It typically contains the following operations: blind rotation (BR), sample extraction (SE), key switching, and modulus switching (MS).

As the goal of bootstrapping is to refresh the noise in the ciphertext, it is necessary to pay extra attention and precisely control the noise generated in each step of the bootstrapping algorithm itself. The basic strategy is to execute the BR step, which mainly generates the new noise, under a large modulus. Then recovering the LWE ciphertext form through SE, reducing the modulus while eliminating the blind rotation noise size through MS, and recovering the original key through key switching algorithm. We introduce two typical bootstrapping work flows as follows:

GINX Bootstrapping [5,6,7]. $\mathrm{LWE}_{571,2^{11}} \xrightarrow{\mathsf{BR}} \mathrm{RLWE}_{1024,2^{25}} \xrightarrow{\mathsf{SE}} \mathrm{LWE}_{1024,2^{25}} \xrightarrow{\mathsf{MS}} \mathrm{LWE}_{1024,2^{14}} \xrightarrow{\mathsf{LtL}} \mathrm{LWE}_{571,2^{14}} \xrightarrow{\mathsf{MS}} \mathrm{LWE}_{571,2^{11}}$.

Remark 3. The above parameters are taken from Lee's recent article [21], with a security level of 128-bit. Due to the update of attack methods, the security level of the parameters in TFHE articles [5,6,7] has been reduced to 115-bit.

LFHE Bootstrapping [18]. $\mathrm{LWE}_{571,2^{11}} \xrightarrow{\mathsf{BR}} \mathrm{RLWE}_{2048,2^{54}} \xrightarrow{\mathsf{MS}} \mathrm{RLWE}_{2048,2^{27}} \xrightarrow{\mathsf{RtR}} \mathrm{RLWE}_{1024,2^{27}} \xrightarrow{\mathsf{SE}} \mathrm{LWE}_{1024,2^{27}} \xrightarrow{\mathsf{MS}} \mathrm{LWE}_{1024,2^{14}} \xrightarrow{\mathsf{LtL}} \mathrm{LWE}_{571,2^{14}} \xrightarrow{\mathsf{MS}} \mathrm{LWE}_{571,2^{11}}$.

To display the switching of the keys and modulus, we use the form $(\mathrm{R})\mathrm{LWE}_{n,q}$ to represent ciphertexts, where n is the dimension of the secret key vector (or polynomial) and q represents the modulus of the ciphertext. LtL stands for LWE to LWE key switching, and both of the above bootstrapping algorithms use its storage version (for a summary and comparison between different versions, see sec. 4). RtR stands for RLWE to RLWE key switching.

3 Gadget Products

The gadget product is used to calculate the scalar multiplication in FHEW-like cryptosystem. It works by gadget decomposing the plaintext scalar and then multiplying the corresponding gadget (R)LWE ciphertexts. This algorithm can reduce the noise growth of scalar multiplication and is widely used in core algorithms such as external product [7] and key switching. This section summarizes three types of gadget products, and analyze their differences in terms of noise growth, auxiliary input size and computational complexity. The first one is the canonical gadget product primarily used for external product. It was first abstracted as a separate algorithm by Micheli et al. in 2023 [9].

Gadget Product: The canonical gadget product $\odot : \mathbb{Z} \times (\mathrm{R})\mathrm{LWE}' \to (\mathrm{R})\mathrm{LWE}$ is defined as:

$$
\begin{aligned}
t \odot \text{(R)LWE}'_{\mathbf{sk}}(\mathbf{m}) &:= \sum_{i=0}^{l-1} t_i \cdot \text{(R)LWE}_{\mathbf{sk}}\left(v_i \cdot \mathbf{m}\right) \\
&= \text{(R)LWE}_{\mathbf{sk}}\left(\sum_{i=0}^{l-1} v_i \cdot t_i \cdot \mathbf{m}\right) \\
&= \text{(R)LWE}_{\mathbf{sk}}\left(t \cdot \mathbf{m} + \varepsilon_{\mathsf{gadget}}(t) \cdot \mathbf{m}\right),
\end{aligned}
$$

Lemma 1. *[18] Let B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the gadget product is bounded by*

$$
\sigma^2_{\odot,\mathsf{input}} \leq \frac{1}{12} l B^2 \sigma^2_{\mathsf{input}} + \frac{1}{3} \mathsf{Var}(\mathbf{m}) \epsilon^2
$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input LWE$'$ *ciphertext, and* $\mathsf{Var}(\mathbf{m})$ *is the variance of the message* $\mathbf{m}$.

Lemma.1 is derived from [18] proposition.1 with the fact $\epsilon \leq \frac{1}{2}\lceil \frac{q}{B^l} \rceil$. This method use the modular multiplication to compute the gadget product. However, for a fixed input $\text{(R)LWE}'_{\mathbf{sk}}(\mathbf{m})$, there is an time-space trade-off that reduces the computational complexity by using additional storage. Specifically, since the range of t_i is bounded by the gadget base B, one can pre-compute and store all possible values of $\text{(R)LWE}'_{\mathbf{sk}}(v_i \cdot t_i \cdot \mathbf{m})$, then use modular addition instead of modular multiplication. This method was first used in the FHEW bootstrapping algorithm proposed by Ducas et al. in 2015 [12], which inspired us to summarize a store version of the gadget product. We denote this method using operator $\oplus$:

Gadget Product (Store Version): The store version $\oplus : \mathbb{Z} \times \text{(R)LWE}' \rightarrow \text{(R)LWE}$ is defined as:

$$
\begin{aligned}
t \oplus \text{(R)LWE}'_{\mathbf{sk}}(\mathbf{m}) &:= \sum_{i=0}^{l-1} \text{(R)LWE}'_{\mathbf{sk}}(v_i \cdot t_i \cdot \mathbf{m}) \\
&= \text{(R)LWE}_{\mathbf{sk}}\left(\sum_{i=0}^{l-1} v_i \cdot t_i \cdot \mathbf{m}\right) \\
&= \text{(R)LWE}_{\mathbf{sk}}\left(t \cdot \mathbf{m} + \varepsilon_{\mathsf{gadget}}(t) \cdot \mathbf{m}\right),
\end{aligned}
$$

Corollary 1. *Let B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the gadget product (store version) is bounded by*

$$
\sigma^2_{\oplus,\mathsf{input}} \leq l \sigma^2_{\mathsf{input}} + \frac{1}{3} \mathsf{Var}(\mathbf{m}) \epsilon^2
$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input LWE$'$ *ciphertext, and* $\mathsf{Var}(\mathbf{m})$ *is the variance of the message* $\mathbf{m}$.

The store version of gadget product use l times modular addition instead of modular multiplication. Thus corollary.1 can be directly derived from lemma.1 by replacing multiplication error growth with addition error growth.

Lastly, we introduce the ring version of the gadget product, denoted by $\odot_R$:

Gadget Product (Ring Version): The Ring gadget product $\odot_R : \mathcal{R} \times \mathrm{RLWE}' \to \mathrm{RLWE}$ is defined as:

$$\begin{aligned}\mathbf{t} \odot_R \mathrm{RLWE}'_{\mathbf{sk}}(\mathbf{m}) &:= \sum_{i=0}^{l-1} \mathbf{t}_i \cdot \mathrm{RLWE}_{\mathbf{sk}}\left(v_i \cdot \mathbf{m}\right) \\ &= \mathrm{RLWE}_{\mathbf{sk}}\left(\sum_{i=0}^{l-1} v_i \cdot \mathbf{t}_i \cdot \mathbf{m}\right) \\ &= \mathrm{RLWE}_{\mathbf{sk}}\left(\mathbf{t} \cdot \mathbf{m} + \varepsilon_{\mathsf{gadget}}(\mathbf{t}) \cdot \mathbf{m}\right),\end{aligned}$$

Corollary 2. *Let n denote the dimension of the ring polynomial of RLWE ciphertexts, B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the gadget product is bounded by*

$$\sigma^2_{\odot_R,\mathsf{input}} \leq \frac{1}{12} nlB^2\sigma^2_{\mathsf{input}} + \frac{1}{3} n\mathsf{Var}(\mathbf{m})\epsilon^2$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input RLWE' *ciphertext, and $\mathsf{Var}(\mathbf{m})$ is the variance of $\mathbf{m}$.*

This algorithm is a ring version of the gadget product. Notice that since polynomial dimension n causes an exponential increase in polynomial gadget decomposition results, it is impractical to accelerate computation by pre-storing all possible $\mathrm{RLWE}'_{\mathbf{sk}}(v_i \cdot \mathbf{t}_i \cdot \mathbf{m})$. In other words, the store version of the ring gadget product is not practical and we do not consider it. The error growth of the ring gadget product needs to take into account the expansion factor of the ring. In this paper, we use a power-of-two cyclotomic ring, with an expansion factor of $\sqrt{n}$ for the two-norm, resulting in a factor of n when evaluating the noise variance. Then corollary.2 can be derived from lemma.1.

Comparison. The computational complexity and auxiliary input size of the three gadget products are listed in tab.2. From lemma.1 and the corollaries in this section, we can conclude that in terms of error growth, $\odot_R = \odot > \oplus$. From tab.2, it is evident that in terms of computational complexity, $\odot_R > \odot > \oplus$, in terms of the size of auxiliary inputs, $\oplus > \odot_R = \odot$.

As the key switching algorithm is always a combination of scalar multiplication and addition, these algorithms can be re-written using the three types of gadget products. This novel perspective makes it easier to examine key switching algorithms, provides insights into their comparison in terms of correctness, error growth, computational complexity, and key size. Our analysis can serve as a guideline for time-space trade-offs in the implementation of the key switching.

Calculation	Computation Complexity	Auxiliary Input (in (R)LWE′)
$\odot$	ln MM	1
$\oplus$	l MA	B
$\odot_R$	l NTT+ln MM	1

Table 2. Comparison between different version of the gadget product, where l and B are the gadget length and base, MA and MM denote the modular addition and modular multiplication operations. NTT is the number theoretic transform algorithm (with $O(ln \log n)$ MM computational complexity) used in polynomial multiplication.

We then revisit LWE-to-LWE key switching algorithms in sec.4, and RLWE-to-RLWE key switching algorithms in sec.5.

4 LWE-to-LWE Key Switching

Chillotti et al. proposed in the TFHE series [5,6,7] that their key switching algorithm can calculate a R-Lipschitz linear morphism while switching keys. This section generalizes all existing LWE-to-LWE key switching algorithms into functional versions, and classifies key switching algorithms into public functional key switching and private functional key switching (following Chilloti et al. [7]) based on whether the Lipschitz morphism needs to be kept confidential.

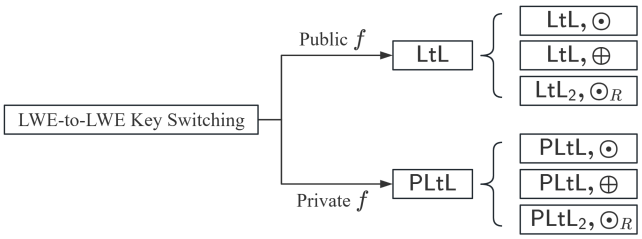

Fig. 2. Six LWE-to-LWE key switching algorithms revisited in this section.

4.1 Public Functional Key Switching

LWE-to-LWE Using Canonical Gadget Product.

- Input: $\mathrm{LWE}_{\mathbf{sk}}(m) = (\mathbf{a}, b)$, and a public R-Lipschitz linear morphism $f : \mathbb{Z} \to \mathbb{Z}$
- Switching key:$\mathsf{LtLK} = \mathrm{LWE}'_{\mathbf{sk}'}(sk_i)_{i\in[1,n]}$
- Output: $\mathrm{LWE}_{\mathbf{sk}'}(f(m))$

– Algorithm:

$$\mathsf{LtL}^{f}_{\mathbf{sk}\to\mathbf{sk}'}(\mathrm{LWE}_{\mathbf{sk}}(m)) := \sum_{i=1}^{n} f(a_i) \odot \mathrm{LWE}'_{\mathbf{sk}'}(sk_i) + (0, f(b)).$$

This algorithm was first proposed by Chillotti et al. [7], and we formalize it using gadget product. We then re-analyze the error growth of this algorithm, and update the theorem 4.1 in [7] for two reasons.

Firstly, TFHE used binary gadget decomposition for scalars in the key switching algorithm. But currently FHEW-like cryptosystems generally use the standard approximate gadget decomposition (power-of-B), as what we considered. Secondly, TFHE utilized the torus algebraic structure in its theoretical analysis, rather than the power of 2 cyclotomic ring used in the implementation. Thus it did not consider the coefficient $1/12$ when calculating the variance of the uniform distribution, resulting in a less compact error bound in theorem 4.1 [7].

Correctness and error analysis:

Theorem 1. *Let n denote the dimension of the LWE ciphertexts, B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the LWE to LWE public functional key switching algorithm is bounded by:*

$$\sigma^2_{\mathsf{LtL}} \leq \frac{1}{12} nlB^2\sigma^2_{\mathsf{LtLK}} + \frac{1}{6} n\epsilon^2 + R^2\sigma^2_{\mathsf{input}},$$

where ϵ is the gadget decomposition error, $\sigma^2_{\mathsf{input}}$ is the error variance of the input LWE ciphertext, and σ^2_{LtLK} is the error variance of the switching key.

Proof. Basing the correctness of the gadget product, we have,

$$\begin{aligned}
&\sum_{i=1}^{n} f(a_i) \odot \mathrm{LWE}'_{\mathbf{sk}'}(sk_i) + (0, f(b)) \\
&= \mathrm{LWE}_{\mathbf{sk}'}\left(\sum_{i=1}^{n} f(a_i \cdot sk_i) + f(b)\right) \\
&= \mathrm{LWE}_{\mathbf{sk}'}\left(f(m) + f(e)\right),
\end{aligned}$$

then we measure the error variance based on lemma.1:

$$\sigma^2_{\mathsf{LtL}} = n\sigma^2_{\odot,\mathsf{LtLK}} + \mathsf{Var}(f(e)) \quad \leq \frac{1}{12} nlB^2\sigma^2_{\mathsf{LtLK}} + \frac{1}{6} n\epsilon^2 + R^2\sigma^2_{\mathsf{input}}.$$

Store Version. As we analyzed in sec.3, the LtL algorithm, which uses the canonical gadget product, also has a corresponding store version. It only requires modifications to the auxiliary input and calculation method:

- Switching key:$\mathsf{LtLK} = \mathrm{LWE}'_{\mathbf{sk}'}(j \cdot sk_i)_{i\in[1,n],j\in[0,B-1]}$
- Algorithm:

$$\mathsf{LtL}^{f}_{\mathbf{sk}\to\mathbf{sk}'}(\mathrm{LWE}_{\mathbf{sk}}(m)) := \sum_{i=1}^{n} f(a_i) \oplus \mathrm{LWE}'_{\mathbf{sk}'}(sk_i) + (0, f(b)).$$

It can be derived from tab.2 that, the store version is faster and has smaller noise growth compared to the canonical LtL algorithm. However, the trade-off is an increase in key size by a factor of B, where B is the base for gadget decomposition.

LWE-to-LWE Using Ring Gadget Product.

- Input: $\mathrm{LWE}_{\vec{sk}}(m) = (\vec{a}, b)$, and a public R-Lipschitz linear morphism $f : \mathbb{Z} \to \mathbb{Z}$
- Switching key: $\mathsf{LtL_2K} = \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk})$, where $\mathbf{sk} = \sum_{i=0}^{l-1} sk_i X^{-i}$, $\mathbf{sk}' = \sum_{i=0}^{l-1} sk'_i X^{-i}$
- Output: $\mathrm{LWE}_{\vec{sk}'}(f(m)) = (\vec{a}', b')$
- Algorithm:

$$(\mathbf{a}', \mathbf{b}') := \sum_{i=1}^{n} f(a_i)X^i \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk}) + (0, f(b)),$$

$$\mathsf{LtL_2}^{f}_{\vec{sk}\to\vec{sk}'}(\mathrm{LWE}_{\vec{sk}}(m)) := (a'_0, a'_1, ..., a'_{n-1}, b'_0).$$

Remark 4. This algorithm involves the conversion between vectors and polynomials. Thus to avoid confusion, we use $\vec{a}$ to represent vectors in this algorithm, while $\mathbf{a}$ to represent polynomials. The notation a_i is the i-th term of the vector $\vec{a}$, and $[\mathbf{a}]_i$ is the i-th coefficient of the polynomial $\mathbf{a}$.

This switching method was proposed by Chen et al. [3]. We formalize it using ring gadget product, and first extend it to the functional version. Therefore, the error growth of this algorithm must take into account the Lipschitz morphism. In addition, Chen et al. only considered the exact gadget decomposition, which is a special case ($q \le B^l$) of the canonical approximate gadget decomposition we use. This also prompts us to re-analyze the error.

Correctness and error analysis:

Theorem 2. *Let n denote the dimension of the LWE ciphertexts, B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the LWE to LWE using* RtR *algorithm is bounded by:*

$$\sigma^2_{\mathsf{LtL_2}} \le \frac{1}{12} nlB^2\sigma^2_{\mathsf{LtL_2K}} + \frac{1}{6}n\epsilon^2 + R^2\sigma^2_{\mathsf{input}},$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input LWE ciphertext, and $\sigma^2_{\mathsf{LtL_2K}}$ is the error variance of the switching key.

Method	Computation complexity	Key size (in bits)
$\mathsf{LtL}, \odot$	$O(ln^2)$ MM	$ln(n+1)\log q$
$\mathsf{LtL}, \oplus$	$(ln+n+1)$ MA	$Bln(n+1)\log q$
$\mathsf{LtL}_2, \odot_R$	$O(ln\log n)$ MM	$2ln\log q$

Table 3. Comparison between different version of the public LWE-to-LWE key switching, where q is the ciphertext modulus, l and B are the gadget length and base, MA and MM denote the modular addition and modular multiplication operations.

Proof. Basing the correctness of the ring gadget product, we have,

$$\begin{aligned} b'_0 + \sum_{i=1}^{n} a'_i sk'_i &= [\mathbf{b}' + \mathbf{a}' \cdot \mathbf{sk}']_0 \\ &= \sum_{i=1}^{n} f(a_i \cdot sk_i) + f(b) \\ &= f(m) + f(e), \end{aligned}$$

thus $(a'_0, a'_1, ..., a'_{n-1}, b'_0)$ is the LWE ciphertext of $f(m)$ under secret key $\vec{sk}'$, then we measure the error variance based on corollary.2:

$$\sigma^2_{\mathsf{LtL}_2} = \sigma^2_{\odot_R, \mathsf{LtL}_2\mathsf{K}} + \mathsf{Var}(f(e)) \quad \leq \frac{1}{12} nlB^2 \sigma^2_{\mathsf{LtLK}} + \frac{1}{6} n\epsilon^2 + R^2 \sigma^2_{\mathsf{input}}$$

Comparison. The computational complexity and key size of LWE-to-LWE public functional key switching algorithms are listed in tab.3. From the theorems in this section, we can conclude that in terms of error growth, we have $(\mathsf{LtL}, \odot) = (\mathsf{LtL}_2, \odot_R) > (\mathsf{LtL}, \oplus)$. From tab.2, it is evident that $(\mathsf{LtL}, \odot) > (\mathsf{LtL}_2, \odot_R) > (\mathsf{LtL}, \oplus)$ in computational complexity. In terms of the key size, we have $(\mathsf{LtL}, \oplus) > (\mathsf{LtL}, \odot) > (\mathsf{LtL}_2, \odot_R)$.

Comparison results indicate that $(\mathsf{LtL}, \odot)$ is inferior to $(\mathsf{LtL}_2, \odot_R)$ in all aspects. Thus when we care more about the computational efficiency and error control of the algorithm, $(\mathsf{LtL}, \oplus)$ is the best choice. On the other hand, if key size (which affects transfer size and storage space) is of greater concern, we should use $(\mathsf{LtL}_2, \odot_R)$ as the substitute.

4.2 Private Functional Key Switching

In the public functional key switching algorithm, the Lipschitz morphism f is used as a public input. However, f should be kept confidential in some cases. For example, it is related to the secret key or derived from a protected model. Chillotti et al. proposed private functional key switching algorithm for this situation [7], where the morphism f is secretly encoded within the algorithm's switching key. In this section, we first revisit this canonical algorithm. Then we

introduce two novel algorithms. The first is the store version of private functional key switching, which we extended based on the method of Ducas et al. [12]. The second is the ring version, extended based on Chen et al.'s methods [3].

Private LWE-to-LWE Using Canonical Gadget Product.

- Input: $\mathrm{LWE}_{\mathbf{sk}}(m) = (\mathbf{a}, b)$
- Switching key: $\mathsf{PLtLK} = (\mathrm{LWE}'_{\mathbf{sk}'}(f(sk_i))_{i\in[1,n]}, \mathrm{LWE}'_{\mathbf{sk}'}(f(1)))$, where $f : \mathbb{Z} \to \mathbb{Z}$ is a private R-Lipschitz linear morphism
- Output: $\mathrm{LWE}_{\mathbf{sk}'}(f(m)) = (\mathbf{a}', b')$
- Algorithm:

$$\mathsf{PLtL}^{f}_{\mathbf{sk}\to\mathbf{sk}'}(\mathrm{LWE}_{\mathbf{sk}}(m)) := \sum_{i=1}^{n} a_i \odot \mathrm{LWE}'_{\mathbf{sk}'}(f(sk_i)) + b \odot \mathrm{LWE}'_{\mathbf{sk}'}(f(1)).$$

Store Version. This version only requires modifications to the switching key and calculation method:

- Switching key:$\mathsf{PLtLK} = (\mathrm{LWE}'_{\mathbf{sk}'}(j \cdot f(sk_i)), \mathrm{LWE}'_{\mathbf{sk}'}(j \cdot f(1)))$, where $i \in [1, n]$, $j \in [0, B-1]$, $f : \mathbb{Z} \to \mathbb{Z}$ is a private R-Lipschitz linear morphism
- Algorithm:

$$\mathsf{PLtL}^{f}_{\mathbf{sk}\to\mathbf{sk}'}(\mathrm{LWE}_{\mathbf{sk}}(m)) := \sum_{i=1}^{n} a_i \oplus \mathrm{LWE}'_{\mathbf{sk}'}(f(sk_i)) + b \oplus \mathrm{LWE}'_{\mathbf{sk}'}(f(1)).$$

Private LWE-to-LWE Using Ring Gadget Product.

- Input: $\mathrm{LWE}_{\vec{sk}}(m) = (\vec{a}, b)$
- Switching key: $\mathsf{LtL_2K} = (\mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk}), \mathrm{RLWE}'_{\mathbf{sk}'}(f(1)))$, where $\mathbf{sk} = \sum_{i=0}^{l-1} f(sk_i)X^{-i}$, $\mathbf{sk}' = \sum_{i=0}^{l-1} sk'_i X^{-i}$, $f : \mathbb{Z} \to \mathbb{Z}$ is a private R-Lipschitz linear morphism
- Output: $\mathrm{LWE}_{\vec{sk}'}(f(m)) = (\vec{a}', b')$
- Algorithm:

$$(\mathbf{a}', \mathbf{b}') := \sum_{i=1}^{n} a_i X^i \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk}) + b \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(f(1)),$$

$$\mathsf{LtL_2}^{f}_{\vec{sk}\to\vec{sk}'}(\mathrm{LWE}_{\vec{sk}}(m)) := (a'_0, a'_1, ..., a'_{n-1}, b'_0).$$

Method	Computation complexity	Key size (in bits)
$\mathsf{PLtL}, \odot$	$O(ln^2)$ MM	$l(n+1)^2 \log q$
$\mathsf{PLtL}, \oplus$	$(ln + n + l + 1)$ MA	$Bl(n+1)^2 \log q$
$\mathsf{PLtL}_2, \odot_R$	$O(ln \log n)$ MM	$2l(n+1) \log q$

Table 4. Comparison between different versions of the private LWE-to-LWE key switching, where q is the ciphertext modulus, l and B are the gadget length and base, MA and MM denote the modular addition and modular multiplication operations.

Correctness, Error Growth and Comparison. The correctness and error analysis of these algorithms are similar to those in section 4.1. For self completeness, we include them in A.1. The computational complexity and key size of LWE-to-LWE private functional key switching algorithms are listed in tab.4.

A comparison of tab.3 and tab.4 reveals that the computational complexity and key size of private algorithms are both larger than the corresponding public algorithms. The comparison results between these three methods are similar to those in sec.4.1: $(\mathsf{PLtL}, \oplus)$ is more suitable for computation-priority scenarios, while $(\mathsf{PLtL}_2, \odot_R)$ is more suitable for storage-priority scenarios.

5 RLWE-to-RLWE Key Switching

Besides LWE-to-LWE key switching, LWE-to-RLWE and RLWE-to-RLWE key switching are also largely described in the literature. However, LWE-to-RLWE algorithms are highly similar to LWE-to-LWE algorithms. Therefore, we put the whole section in the Appendix A.3 for readers to refer to the algorithms and theorems. RLWE-to-RLWE key switching is different. To the best of our knowledge, it can only be calculated through ring gadget product.

In this section, we extend this method into functional versions, which support calculation of both public and private Lipschitz functions. We also prove that the widely-used scheme switching algorithm [9] (or EvalSquareMult algorithm [18]) is a special case of our extended private functional key switching algorithm.

5.1 Public Functional Key Switching

RLWE-to-RLWE Using Ring Gadget Product.

- Input: $\mathrm{RLWE}_{\mathbf{sk}}(\mathbf{m}) = (\mathbf{a}, \mathbf{b})$, and a public R-Lipschitz linear morphism $f : \mathcal{R} \to \mathcal{R}$
- Switching key: $\mathsf{RtRK} = \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk})$
- Output: $\mathrm{RLWE}_{\mathbf{sk}'}(f(\mathbf{m})) = (\mathbf{a}', \mathbf{b}')$
- Algorithm:

$$\mathsf{RtR}_{\mathbf{sk}\to\mathbf{sk}'}(\mathrm{RLWE}_{\mathbf{sk}}(\mathbf{m})) := f(\mathbf{a}) \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk}) + (0, f(\mathbf{b})).$$

Correctness and error analysis:

Theorem 3. *Let n denote the dimension of the ring polynomial of RLWE ciphertexts, B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the LWE to LWE public functional key switching algorithm is bounded by:*

$$\sigma^2_{\mathsf{RtR}} \leq \frac{1}{12} nlB^2 \sigma^2_{\mathsf{RtRK}} + \frac{1}{6} n\epsilon^2 + \sigma^2_{\mathsf{input}},$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input RLWE ciphertext, and σ^2_{RtLR} is the error variance of the switching key.

Proof. Basing the correctness of the Ring gadget product, we have,

$$\begin{aligned} & f(\mathbf{a}) \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk}) + (0, f(\mathbf{b})) \\ = \; & \mathrm{RLWE}_{\mathbf{sk}'}(f(\mathbf{a} \cdot \mathbf{sk}) + f(\mathbf{b})) \\ = \; & \mathrm{RLWE}_{\mathbf{sk}'}(f(\mathbf{m}) + f(\mathbf{e})). \end{aligned}$$

then we measure the error variance based on lemma.2:

$$\sigma^2_{\mathsf{RtR}} = \sigma^2_{\odot_R, \mathsf{RtRK}} + \mathsf{Var}(\mathbf{e}) \quad \leq \frac{1}{12} nlB^2 \sigma^2_{\mathsf{RtRK}} + \frac{1}{6} n\epsilon^2 + R^2 \sigma^2_{\mathsf{input}}.$$

5.2 Private Functional Key Switching

Private RLWE-to-RLWE Using Ring Gadget Product.

- Input: $\mathrm{RLWE}_{\mathbf{sk}}(\mathbf{m}) = (\mathbf{a}, \mathbf{b})$
- Switching key: $\mathsf{RtRK} = (\mathrm{RLWE}'_{\mathbf{sk}'}(f(\mathbf{sk})), \mathrm{RLWE}'_{\mathbf{sk}'}(f(1))$, where $f : \mathcal{R} \to \mathcal{R}$ is a private R-Lipschitz linear morphism
- Output: $\mathrm{RLWE}_{\mathbf{sk}'}(f(\mathbf{m})) = (\mathbf{a}', \mathbf{b}')$
- Algorithm:

$$\mathsf{RtR}_{\mathbf{sk} \to \mathbf{sk}'}(\mathrm{RLWE}_{\mathbf{sk}}(\mathbf{m})) := \mathbf{a} \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(f(\mathbf{sk})) + \mathbf{b} \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(f(1)).$$

The correctness and error analysis of this algorithm is similar to theorem.3. We put it in A.2 for self completeness. When the private Lipschitz morphism is $f(x) = \mathbf{sk} \cdot x$, our algorithm becomes: $\mathbf{a} \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk}^2)) + \mathbf{b} \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk}) = \mathbf{a} \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk}^2)) + (\mathbf{b}, 0)$, where the right side is the well-known scheme switching algorithm [9] (or EvalSquareMult algorithm [18]).

6 Light-Key Bootstrapping

To illustrate the effectiveness of our result, we apply the above analysis to construct light-key bootstrapping algorithms. First, we modify the classical GINX bootstrapping [7], for which our method provides a time-space trade-off. We then improve the LFHE (light-key FHE) bootstrapping proposed by Kim et al. [18], which is specifically designed to reduce the key size. We optimize their result and yield the bootstrapping algorithm with the smallest key.

Parameters	Q	Q_{RtR}	Q_{LtL}	q	N	N_{RtR}	n	l_{br}	l_{ak}	l_{sqk}	l_{RtR}	l_{LtL}
GINX [21]	25	–	14	11	1024	–	571	4	–	–	–	2
GINX_our	25	–	15	11	1024	–	571	4	–	–	–	13
LFHE [18]	54	27	14	11	2048	1024	571	3	5	2	2	3
LFHE_our	54	27	15	11	2048	1024	571	3	5	2	2	13

Table 5. Security and Parameters.

6.1 Security and Parameters

GINX bootstrapping algorithm use $(\mathsf{LtL}, \oplus)$ for key switching due to its higher efficiency and lower noise growth. However, the large key size of $(\mathsf{LtL}, \oplus)$ results in the key switching key occupying 91.6 % of the GINX bootstrapping key (see tab.1). LFHE replace part of the key switching from $(\mathsf{LtL}, \oplus)$ to $(\mathsf{RtR}, \odot_R)$, which has a smaller key size. However, it still retains an $(\mathsf{LtL}, \oplus)$ step, so that the key switching key still occupies 48.3 % of the LFHE bootstrapping key.

In order to construct light-key bootstrapping algorithms, our idea is to use $(\mathsf{LtL}_2, \odot_R)$ to replace $(\mathsf{LtL}, \oplus)$ in GINX and LFHE bootstrapping. However, a direct adoption would not work since the noise growth of $(\mathsf{LtL}_2, \odot_R)$ is much higher than that of $(\mathsf{LtL}, \oplus)$ under the same parameters. Therefore, to ensure algorithm security and control noise introduced by bootstrapping itself, we made necessary adjustments to the bootstrapping parameters, see tab.6.1. This set of parameters ensures that the security level of algorithms exceeds 128-bit [5], and the decryption failure rate due to noise accumulation is less than 2^{-32} [6].

q and n denotes the modulus and dimension of the ciphertext before bootstrapping. Q, N, l_{br}, l_{ak}, and l_{sqk} are the parameters used for blind rotation, representing the modulus and ring dimension of the blind rotation key, and the gadget length in blind rotation, automorphism, and SquareKeyMult (the latter two are used in LFHE), respectively. Q_{RtR}, N_{RtR}, and l_{RtR} denote the modulus, ring dimension, and gadget decomposition length of the RtR key switching key. Q_{LtL} and l_{LtL} represent the modulus and gadget decomposition length of the LtL_2 key switching key.

6.2 Work Flow

The improved GINX and LFHE bootstrapping are shown as follows (all abbreviations are defined in previous section, or check sec.2.4 for explanations):

$$\mathbf{GINX_{our}}: \mathrm{LWE}_{571,2^{11}} \xrightarrow{\mathsf{BR}} \mathrm{RLWE}_{1024,2^{25}} \xrightarrow{\mathsf{SE}} \mathrm{LWE}_{1024,2^{25}} \xrightarrow{\mathsf{MS}} \mathrm{LWE}_{1024,2^{15}} \xrightarrow{\mathsf{LtL}_2} \mathrm{LWE}_{571,2^{15}} \xrightarrow{\mathsf{MS}} \mathrm{LWE}_{571,2^{11}}$$

$$\mathbf{LFHE_{our}}: \mathrm{LWE}_{571,2^{11}} \xrightarrow{\mathsf{BR}} \mathrm{RLWE}_{2048,2^{54}} \xrightarrow{\mathsf{MS}} \mathrm{RLWE}_{2048,2^{27}} \xrightarrow{\mathsf{RtR}} \mathrm{RLWE}_{1024,2^{27}} \xrightarrow{\mathsf{SE}} \mathrm{LWE}_{1024,2^{27}} \xrightarrow{\mathsf{MS}} \mathrm{LWE}_{1024,2^{15}} \xrightarrow{\mathsf{LtL}_2} \mathrm{LWE}_{571,2^{15}} \xrightarrow{\mathsf{MS}} \mathrm{LWE}_{571,2^{11}}$$

[5] test by LWE estimator, https://bitbucket.org/malb/lwe-estimator/src/master/

[6] calculate by $1 - \mathrm{erf}\left(\frac{q}{8\sqrt{2}\sigma}\right)$, where erf represents the Gaussian error function.

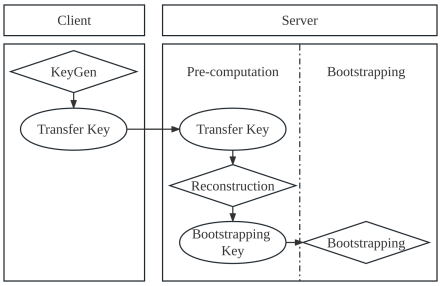

Fig. 3. The transfer model of TFHE bootstrapping.

6.3 Key Size

This section analyze the bootstrapping key size. Since the modulus switching and sample extraction algorithms do not require evaluation keys, the bootstrapping key includes two parts: the blind rotation key and the key switching key.

GINX$_{\text{our}}$: The blind rotation key contains n RGSW ciphertexts, with each having $4l_{\mathsf{br}}N\log Q$ bits. This results in a blind rotation key size of 27.88 MB. We use LtL_2 instead of LtL_1, the key contains 1 RLWE$'$ ciphertext with a size of $2nl_{\mathsf{LtL}}\log Q_{\mathsf{LtL}}$ bits. Thus the key switching key size is 27.2 KB. The total key size is 27.91 MB.

LFHE$_{\text{our}}$: The blind rotation key also contains n RGSW ciphertexts, resulting in a blind rotation key size of 90.33 MB. The RtR key switching key contains 1 RLWE$'$ ciphertext with a size of $2Nl_{\mathsf{RtR}}\log Q_{\mathsf{RtR}}$ bits, resulting in a key size of 27 KB. The LtL_2 key switching key size is 27.2 KB. The total is 90.38 MB.

6.4 Transfer Model and Transfer Key Size

LFHE [18] proposed a transfer model, see fig.3. The client transmits a transfer key (a seed) to the server. Then the server runs the reconstruction algorithm to obtain the complete bootstrapping key, and performs the bootstrapping algorithm. In this model, client and server utilize a common reference string (CRS) to generate the **a**-components of each transferred LWE and RLWE ciphertext. Thus only the b(or **b** for RLWE)-components of the ciphertext needs to be transmitted. LFHE's blind rotation algorithm is specifically designed for the transfer model and uses a pached blind rotation key to reduce the bootstrapping transfer key size to within 1 MB. We also calculate the transfer key size of our improved algorithms under the transfer model.

GINX$_{\text{our}}$: The blind rotation key contains n RGSW ciphertexts, with each needing to transfer $2l_{\mathsf{br}}N\log Q$ bits. This results in a blind rotation transfer key

Methods	Transfer key size	Bootstrapping key size
GINX	16.48 MB	250 MB
GINX_{our}	13.96 MB	27.91 MB
LFHE	881 KB	175 MB
LFHE_{our}	810.1 KB	90.38 MB

Table 6. Transfer key size and the bootstrapping key size in different methods.

size of 13.94 MB. We use LtL_2 instead of LtL_1, the key contains 1 RLWE$'$ ciphertext, with a transfer key size of $nl_{\mathsf{LtL}} \log Q_{\mathsf{LtL}}$ bits. This results in a key switching transfer key size of 13.6 KB. The total is 13.96 MB.

LFHE_{our} : The packed blind rotation key contains 1 RLWE ciphertext and $(\log N+1)$ RLWE$'$ ciphertexts. Each RLWE ciphertext needs to transfer $N \log Q$ bits, each RLWE$'$ ciphertext needs to transfer $Nl_{\mathsf{ak}}(l_{\mathsf{sqk}}) \log Q$ bits. This results in a blind rotation transfer key size of 783 KB. The RtR key switching key contains 1 RLWE$'$ ciphertext and has a transfer size of $Nl_{\mathsf{RtR}} \log Q_{\mathsf{RtR}}$ bits, resulting in a transfer key size of 13.5 KB. The LtL_2 transfer key size is 13.6 KB. The total key size is 810.1 KB.

For GINX bootstrapping, our method reduces the bootstrapping key size by 88.8 % and the transfer key size by 15.3 %. We do not want to oversell this result, but take it as a trade-off method towards practical TFHE applications. For LFHE bootstrapping, our method outperforms Kim's method [18] by reducing 48.4% bootstrapping key size and 8 % transfer key size.

7 Conclusion

The key switching algorithm is crucial in real-world fully homomorphic encryption (FHE) applications due to its significant impact on the key size and efficiency of the FHE system. This paper revisits currently known key switching algorithms, expands their functionality, carefully recalculates the error growth, and provide a comparison of different algorithms under the same benchmark. Our analysis is applied to the bootstrapping algorithm, resulting in optimal light-key FHE. This paper can be served as an reference for the time-space trade-off of key switching algorithms and assists to build FHE applications with different computational and storage requirements.

Acknowledgments

We are grateful for the helpful comments from the anonymous reviewers of ICISC 2023. This work was supported by CAS Project for Young Scientists in Basic Research (Grant No. YSBR-035).

References

1. Amuthan, A., Sendhil, R.: Hybrid gsw and dm based fully homomorphic encryption scheme for handling false data injection attacks under privacy preserving data aggregation in fog computing. Journal of Ambient Intelligence and Humanized Computing **11**, 5217–5231 (2020)
2. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. Journal of Cryptology **31**(3), 885–916 (2018)
3. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient homomorphic conversion between (ring) lwe ciphertexts. In: International Conference on Applied Cryptography and Network Security. pp. 460–479. Springer (2021)
4. 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)
5. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. ADVANCES IN CRYPTOLOGY-ASIACRYPT 2016, PT I **10031**, 3–33 (2016)
6. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. 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. pp. 377–408. Springer (2017)
7. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
8. Cosseron, O., Hoffmann, C., Méaux, P., Standaert, F.X.: Towards case-optimized hybrid homomorphic encryption: Featuring the elisabeth stream cipher. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 32–67. Springer (2022)
9. De Micheli, G., Kim, D., Micciancio, D., Suhl, A.: Faster amortized fhew bootstrapping using ring automorphisms. Cryptology ePrint Archive (2023)
10. Deviani, R.: The application of fully homomorphic encryption on xgboost based multiclass classification. JIEET (Journal of Information Engineering and Educational Technology) **7**(1), 49–58 (2023)
11. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: a cipher with low anddepth and few ands per bit. In: Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38. pp. 662–692. Springer (2018)
12. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34. pp. 617–640. Springer (2015)
13. Gomes, F.A., de Matos, F., Rego, P., Trinta, F.: Analysis of the impact of homomorphic algorithm on offloading of mobile application tasks. In: 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC). pp. 961–962. IEEE (2023)
14. Halevi, S., Shoup, V.: Bootstrapping for helib. Journal of Cryptology **34**(1), 7 (2021)

15. Jiang, L., Lou, Q., Joshi, N.: Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus. In: Proceedings of the 59th ACM/IEEE Design Automation Conference. pp. 235–240 (2022)
16. Jutla, C.S., Manohar, N.: Modular lagrange interpolation of the mod function for bootstrapping of approximate he. Cryptology ePrint Archive (2020)
17. Jutla, C.S., Manohar, N.: Sine series approximation of the mod function for bootstrapping of approximate he. Springer-Verlag (2022)
18. Kim, A., Lee, Y., Deryabin, M., Eom, J., Choi, R.: Lfhe: Fully homomorphic encryption with bootstrapping key size less than a megabyte. Cryptology ePrint Archive (2023)
19. Kocabas, O., Soyata, T.: Towards privacy-preserving medical cloud computing using homomorphic encryption. In: Virtual and Mobile Healthcare: Breakthroughs in Research and Practice, pp. 93–125. IGI Global (2020)
20. 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)
21. Lee, Y., Micciancio, D., Kim, A., Choi, R., Deryabin, M., Eom, J., Yoo, D.: Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 227–256. Springer (2023)
22. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 1–23. Springer, Heidelberg (2010)
23. Micciancio, D., Polyakov, Y.: Bootstrapping in fhew-like cryptosystems. In: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 17–28 (2021)
24. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop. pp. 113–124 (2011)
25. Nam, K., Oh, H., Moon, H., Paek, Y.: Accelerating n-bit operations over tfhe on commodity cpu-fpga. In: Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design. pp. 1–9 (2022)
26. Peralta, G., Cid-Fuentes, R.G., Bilbao, J., Crespo, P.M.: Homomorphic encryption and network coding in iot architectures: Advantages and future challenges. Electronics **8**(8), 827 (2019)
27. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM **56**(6), 34:1–34:40 (2009)
28. Ren, W., Tong, X., Du, J., Wang, N., Li, S.C., Min, G., Zhao, Z., Bashir, A.K.: Privacy-preserving using homomorphic encryption in mobile iot systems. Computer Communications **165**, 105–111 (2021)
29. Shrestha, R., Kim, S.: Integration of iot with blockchain and homomorphic encryption: Challenging issues and opportunities. In: Advances in computers, vol. 115, pp. 293–331. Elsevier (2019)
30. Ye, T., Kannan, R., Prasanna, V.K.: Fpga acceleration of fully homomorphic encryption over the torus. In: 2022 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–7. IEEE (2022)

A Appendix

A.1 LWE-to-LWE Key Switching

Private LWE-to-LWE Using Canonical Gadget Product

Theorem 4. *Let n denote the dimension of the LWE ciphertexts, B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the LWE to LWE public functional key switching algorithm is bounded by:*

$$\sigma^2_{\mathsf{PLtL}} \leq \frac{1}{12}(n+1)lB^2\sigma^2_{\mathsf{LtLK}} + \frac{1}{6}R^2n\epsilon^2 + +\frac{1}{3}n\mathsf{Var}(f(1))\epsilon^2 + R^2\sigma^2_{\mathsf{input}},$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input LWE ciphertext, and σ^2_{LtLK} is the error variance of the switching key.

Proof. Basing the correctness of the gadget product, we have,

$$\begin{aligned}
&\sum_{i=1}^{n} a_i \odot \mathrm{LWE}'_{\mathbf{sk}'}(f(sk_i)) + b \odot \mathrm{LWE}'_{\mathbf{sk}'}(f(1)) \\
&= \mathrm{LWE}_{\mathbf{sk}'}\left(\sum_{i=1}^{n} f(a_i \cdot sk_i) + f(b)\right) \\
&= \mathrm{LWE}_{\mathbf{sk}'}\left(f(m) + f(e)\right),
\end{aligned}$$

then we measure the error variance based on lemma.1:

$$\begin{aligned}
\sigma^2_{\mathsf{PLtL}} &= \sum_{i=1}^{n} \sigma^2_{\odot,\mathrm{LWE}'_{\mathbf{sk}'}(f(sk_i))} + \sigma^2_{\odot,\mathrm{LWE}'_{\mathbf{sk}'}(f(1))} + \mathsf{Var}(f(e)) \\
&\leq \frac{1}{12}(n+1)lB^2\sigma^2_{\mathsf{LtLK}} + \frac{1}{3}n\mathsf{Var}(f(sk_i)\epsilon^2 + \frac{1}{3}n\mathsf{Var}(f(1))\epsilon^2 + R^2\sigma^2_{\mathsf{input}}. \\
&\leq \frac{1}{12}(n+1)lB^2\sigma^2_{\mathsf{LtLK}} + \frac{1}{6}R^2n\epsilon^2 + +\frac{1}{3}n\mathsf{Var}(f(1))\epsilon^2 + R^2\sigma^2_{\mathsf{input}}.
\end{aligned}$$

Private LWE-to-LWE Using Ring Gadget Product

Theorem 5. *Let n denote the dimension of the LWE ciphertexts, B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the LWE to LWE using* RtR *algorithm is bounded by:*

$$\sigma^2_{\mathsf{PLtL}_2} \leq \frac{1}{6}NlB^2\sigma^2_{\mathsf{LtL}_2\mathsf{K}} + \frac{1}{6}R^2n\epsilon^2 + \frac{1}{3}n\mathsf{Var}(f(1))\epsilon^2 + R^2\sigma^2_{\mathsf{input}},$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input LWE ciphertext, and $\sigma^2_{\mathsf{PLtL}_2\mathsf{K}}$ is the error variance of the switching key.

Proof. Basing the correctness of the ring gadget product, we have,

$$b'_0 + \sum_{i=1}^{n} a'_i sk'_i = [\mathbf{b}' + \mathbf{a}' \cdot \mathbf{sk}']_0 \quad = \sum_{i=1}^{n} f(a_i \cdot sk_i) + f(b) = f(m) + f(e),$$

thus $(a'_0, a'_1, ..., a'_{n-1}, b'_0)$ is the LWE ciphertext of $f(m)$ under secret key $\vec{sk}'$, then we measure the error variance based on corollary.2:

$$\begin{aligned}
\sigma^2_{\mathsf{LtL_2}} &= \sigma^2_{\odot_R, \mathrm{RLWE}'_{\mathbf{sk}'}(\mathbf{sk})} + \sigma^2_{\odot_R, \mathrm{RLWE}'_{\mathbf{sk}'}(f(\mathbf{1}))} + \mathsf{Var}(f(\mathbf{e})) \\
&\leq \frac{1}{6} N l B^2 \sigma^2_{\mathsf{LtL_2K}} + \frac{1}{3} n \mathsf{Var}(f(sk_i)\epsilon^2 + \frac{1}{3} n \mathsf{Var}(f(1))\epsilon^2 + R^2 \sigma^2_{\mathsf{input}} \\
&\leq \frac{1}{6} N l B^2 \sigma^2_{\mathsf{LtL_2K}} + \frac{1}{6} R^2 n \epsilon^2 + \frac{1}{3} n \mathsf{Var}(f(1))\epsilon^2 + R^2 \sigma^2_{\mathsf{input}}.
\end{aligned}$$

A.2 RLWE-to-RLWE Key Switching

Private RLWE-to-RLWE

Theorem 6. *Let n denote the dimension of the ring polynomial of RLWE ciphertexts, B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the LWE to LWE public functional key switching algorithm is bounded by:*

$$\sigma^2_{\mathsf{RtR}} \leq \frac{1}{12} n l B^2 \sigma^2_{\mathsf{RtRK}} + \frac{1}{6} n \epsilon^2 + \sigma^2_{\mathsf{input}},$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input RLWE ciphertext, and σ^2_{RtLR} is the error variance of the switching key.

Proof. Basing the correctness of the Ring gadget product, we have,

$$\begin{aligned}
&\mathbf{a} \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}\left(f(\mathbf{sk})\right) + \mathbf{b} \odot_R \mathrm{RLWE}'_{\mathbf{sk}'}\left(f(1)\right) \\
&= \mathrm{RLWE}_{\mathbf{sk}'}\left(f(\mathbf{a} \cdot \mathbf{sk}) + f(\mathbf{b})\right) \\
&= \mathrm{RLWE}_{\mathbf{sk}'}\left(f(\mathbf{m}) + f(\mathbf{e})\right).
\end{aligned}$$

then we measure the error variance based on lemma.2:

$$\begin{aligned}
\sigma^2_{\mathsf{RtR}} &= \sigma^2_{\odot_R, \mathrm{RLWE}'_{\mathbf{sk}'}(f(\mathbf{sk}))} + \sigma^2_{\odot_R, \mathrm{RLWE}'_{\mathbf{sk}'}(f(\mathbf{1}))} + \mathsf{Var}(f(\mathbf{e})) \\
&\leq \frac{1}{6} n l B^2 \sigma^2_{\mathsf{RtRK}} + \frac{1}{3} n \mathsf{Var}(f(\mathbf{sk}))\epsilon^2 + \frac{1}{3} n \mathsf{Var}(f(1))\epsilon^2 + R^2 \sigma^2_{\mathsf{input}} \\
&\leq \frac{1}{6} n l B^2 \sigma^2_{\mathsf{RtRK}} + \frac{1}{6} R^2 n \epsilon^2 + \frac{1}{3} n \mathsf{Var}(f(1))\epsilon^2 + R^2 \sigma^2_{\mathsf{input}}
\end{aligned}$$

A.3 LWE-to-RLWE Key Switching

Public LWE-to-RLWE Functional Key Switching

Input: $\mathrm{LWE}_{\mathbf{sk}}(m) = (\mathbf{a}, b)$, and a public R-Lipschitz morphism $f : \mathbb{Z} \to \mathbb{Z}$
Switching key: $\mathsf{LtRK} = \mathrm{RLWE}'_{\mathbf{sk}'}(sk_i)_{i\in[1,n]}$
Output: $\mathrm{RLWE}_{\mathbf{sk}'}(f(m)) = (\mathbf{a}', \mathbf{b}')$
Algorithm:

$$\mathsf{LtR}^{f}_{\mathbf{sk}\to\mathbf{sk}'}(\mathrm{LWE}_{\mathbf{sk}}(m)) := \sum_{i=1}^{n} f(a_i) \odot \mathrm{RLWE}'_{\mathbf{sk}'}(sk_i) + (0, f(b)).$$

Correctness and error analysis:

Theorem 7. *Let n denote the dimension of the LWE ciphertexts, B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the LWE to LWE public functional key switching algorithm is bounded by:*

$$\sigma^2_{\mathsf{LtR}} \leq \frac{1}{12} nlB^2\sigma^2_{\mathsf{LtRK}} + \frac{1}{6}n\epsilon^2 + R^2\sigma^2_{\mathsf{input}},$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input LWE ciphertext, and σ^2_{LtLK} is the error variance of the switching key.

Proof. Basing the correctness of the gadget product, we have,

$$\begin{aligned}
&\sum_{i=1}^{n} f(a_i) \odot \mathrm{RLWE}'_{\mathbf{sk}'}(sk_i) + (0, f(b)) \\
&= \mathrm{RLWE}_{\mathbf{sk}'}\left(\sum_{i=1}^{n} f(a_i \cdot sk_i) + f(b)\right) \\
&= \mathrm{RLWE}_{\mathbf{sk}'}\left(f(m) + f(e)\right),
\end{aligned}$$

then we measure the error variance based on lemma.1:

$$\sigma^2_{\mathsf{LtR}} = n\sigma^2_{\odot,\mathsf{LtRK}} + \mathsf{Var}(f(e)) \quad \leq \frac{1}{12} nlB^2\sigma^2_{\mathsf{LtLK}} + \frac{1}{6}n\epsilon^2 + R^2\sigma^2_{\mathsf{input}}$$

Private LWE-to-RLWE Functional Key Switching

Input: $\mathrm{LWE}_{\mathbf{sk}}(m) = (\mathbf{a}, b)$
Switching key: $\mathsf{PLtRK} = (\mathrm{RLWE}'_{\mathbf{sk}'}(f(sk_i))_{i\in[1,n]}, \mathrm{RLWE}'_{\mathbf{sk}'}(f(1)))$, where $f : \mathbb{Z} \to \mathbb{Z}$ is a private R-Lipschitz linear morphism
Output: $\mathrm{RLWE}_{\mathbf{sk}'}(f(m)) = (\mathbf{a}', \mathbf{b}')$
Algorithm:

$$\mathsf{PLtR}^{f}_{\mathbf{sk}\to\mathbf{sk}'}(\mathrm{LWE}_{\mathbf{sk}}(m)) := \sum_{i=1}^{n} a_i \odot \mathrm{RLWE}'_{\mathbf{sk}'}(f(sk_i)) + b \odot \mathrm{RLWE}'_{\mathbf{sk}'}(f(1)).$$

Correctness and error analysis:

Theorem 8. *Let n denote the dimension of the LWE ciphertexts, B and l denote the base and the length of the gadget decomposition, respectively, then the error variance of the result of the LWE to LWE public functional key switching algorithm is bounded by:*

$$\sigma^2_{\mathsf{PLtR}} \leq \frac{1}{12}(n+1)lB^2\sigma^2_{\mathsf{LtLK}} + \frac{1}{6}R^2(n+1)\epsilon^2 + R^2\sigma^2_{\mathsf{input}},$$

where $\sigma^2_{\mathsf{input}}$ is the error variance of the input LWE ciphertext, and σ^2_{LtLK} is the error variance of the switching key.

Proof. Basing the correctness of the gadget product, we have,

$$\begin{aligned}
&\sum_{i=1}^{n} a_i \odot \mathrm{RLWE}'_{\mathbf{sk}'}(f(sk_i)) + b \odot \mathrm{RLWE}'_{\mathbf{sk}'}(f(1)) \\
&= \mathrm{RLWE}_{\mathbf{sk}'}\left(\sum_{i=1}^{n} f(a_i \cdot sk_i) + f(b)\right) \\
&= \mathrm{RLWE}_{\mathbf{sk}'}\left(f(m) + f(e)\right),
\end{aligned}$$

then we measure the error variance based on lemma.1:

$$\begin{aligned}
\sigma^2_{\mathsf{PLtR}} &= (n+1)\sigma^2_{\odot,\mathsf{PLtRK}} + \mathsf{Var}(f(e)) \\
&\leq \frac{1}{12}(n+1)lB^2\sigma^2_{\mathsf{LtLK}} + \frac{1}{6}R^2(n+1)\epsilon^2 + R^2\sigma^2_{\mathsf{input}}.
\end{aligned}$$