# LFHE: Fully Homomorphic Encryption with Bootstrapping Key Size Less than a Megabyte

Andrey Kim[1], Yongwoo Lee*[1,2], Maxim Deryabin[1], Jieun Eom[1], and Rakyong Choi[1]

[1] Samsung Advanced Institute of Technology, Suwon, Republic of Korea
{andrey.kim, max.deriabin, jieun.eom, rakyong.choi}@samsung.com
[2] Inha University, Incheon, Republic of Korea
yongwoo@inha.ac.kr

May 26, 2023

**Abstract**

Fully Homomorphic Encryption (FHE) enables computations to be performed on encrypted data, so one can outsource computations of confidential information to an untrusted party. Ironically, FHE requires the client to generate massive evaluation keys and transfer them to the server side where all computations are supposed to be performed. In this paper, we propose LFHE, the Light-key FHE variant of the FHEW scheme introduced by Ducas and Micciancio in Eurocrypt 2015, and its improvement TFHE scheme proposed by Chillotti et al. in Asiacrypt 2016. In the proposed scheme the client generates small packed evaluation keys, which can be transferred to the server side with much smaller communication overhead compared to the original non-packed variant. The server employs a key reconstruction technique to obtain the evaluation keys needed for computations.

This approach allowed us to achieve the FHE scheme with the packed evaluation key transferring size of less than a Megabyte, which is an order of magnitude improvement compared to the best-known methods.

**Keywords:** Bootstrapping, Fully Homomorphic Encryption.

*A. Kim and Y. Lee contributed equally.

# Contents

# 1 Introduction

After over a decade of intensive research on Fully Homomorphic Encryption (FHE), this concept is gradually becoming more practical. FHE enables privacy-preserving outsourced computations by allowing data to remain encrypted while being processed by an untrusted party. In this scenario, a data owner (*client*) employs encryption to protect its sensitive data before transmitting it to a computing party (*server*). The server can then perform computations on the data without having to decrypt it. FHE originated from various partial homomorphic encryption (HE) schemes after Craig Gentry's whitepaper [Gen09] where he introduced bootstrapping operation to enable arbitrary computations over encrypted data without interactions with a data owner. In recent FHE research, significant advancements have been made in the development of efficient versions of HE schemes tailored to various computing scenarios, including computations with binary, integer, and floating-point numbers. However, real-world systems relying on HE as a key technology for private computations continue to face a number of practical challenges.

One issue that arises in secure computation is the communication overhead between the client and server. Homomorphic encryption schemes use lattice-based encryption, which leads to larger ciphertexts compared to unencrypted messages. To address this issue and reduce the computational and communication overhead for the client, a transciphering technique has been studied in several articles [NLV11, CDKS21, CHK$^+$21, LHH$^+$21, CIL21]. Essentially, this technique involves encrypting the message with a symmetric cipher and then having the server perform homomorphic decryption on the symmetric ciphertext, where the secret key is given in HE-encrypted form. However, the issue of communication overhead between the client and server is not limited to sending encrypted messages. The homomorphic computation also requires the generation of *evaluation keys*, which contain encrypted information about the secret key held by the client. The generation and transmission of evaluation keys can be a resource-intensive process that requires immense communication and storage capabilities. This issue gives rise to another challenge in that the evaluation keys include the keys required for the bootstrapping operation, which are commonly referred to as bootstrapping keys. To overcome this challenge, it is necessary to develop efficient techniques for generating and transmitting evaluation keys.

RLWE-based FHE schemes roughly can be classified as "word encryption" schemes which include BGV [BGV14], BFV [BV11], and CKKS [CKKS17], and FHEW-like "bitwise encryption" schemes such as DM/FHEW [DM15] and CGGI/TFHE [CGGI20]. Bootstrapping for the first type of scheme is resource-intensive. To keep system performance high the client has to generate several gigabytes (GB) of bootstrapping keys [BMTH21, BCC$^+$22]. The issue with the size of bootstrapping keys for the CKKS scheme was addressed in [LLKN22], where the transmit key size is reduced from several dozens to 3.9GB. Despite the fact that RLWE-based schemes such as BGV, BFV, and CKKS enjoy good amortized performance, the size of the bootstrapping keys still remains a challenging issue.

On the other hand, FHEW-like schemes enjoy significantly smaller bootstrapping key sizes, although they can be less efficient at packing. While the original DM/FHEW scheme [DM15], which relies on the AP bootstrapping procedure [AP14], requires a large number of bootstrapping keys (approximately 800 MB for parameters that ensure 128-bit security [LMK$^+$23]), its more recent improvement, the CGGI/TFHE scheme [CGGI17, CGGI20] based on the GINX bootstrapping [GINX16], has reduced this requirement, in part due to the limiting the selection of secret keys. For example, this approach necessitates approximately 20MB and 40MB of bootstrapping keys when the secret key consists of a vector of binary or ternary values, respectively. Recent

studies have demonstrated that efficient bootstrapping can be achieved for arbitrary secret keys with approximately 12MB of bootstrapping keys [LMK+23].

Despite this recent progress, evaluation keys of size 10MB-40MB still limit potential applications of FHEW-like schemes. Note that the evaluation keys of such size are required to perform only a single operation over a small integer or binary value. To extend the scheme for additional operations or increase the size of values, the evaluation key size should be also increased. At the same time, some potential applications such as mobile devices, IoT, and blockchain assume that client devices are very limited in terms of resources [PCFBC19, SK19, GdMRT23, RTD+21]. In such cases, small communication cost is more important than computations on the server side.

### 1.1 Contribution

In this paper, we concentrate on reducing the size of bootstrapping keys required for FHEW-like schemes. First, we propose a general notion of compressing bootstrapping keys by the secret key holder (client) and reconstructing them on the computing party (server) before doing computations. Figure 1 shows a brief sketch of the proposed method.

In more detail, we focus on the CGGI scheme with a binary secret key. We observe that parts of bootstrapping keys for the CGGI scheme consist of lots of RLWE encryptions of a single value, while RLWE ciphertexts are capable to pack a lot of values at the same time. We present an algorithmic approach that has the client pack these values into a small number of RLWE ciphertexts and extracts them as separate RLWE ciphertexts on the server side. The other part of bootstrapping keys can be reconstructed with small additional auxiliary keys. Our approach is developed on the brief sketch which was initially mentioned in the early preprint [KDE+21]. We specify the algorithms and parameters and demonstrate their feasibility by an implementation.

As the key reconstruction introduces additional noise, the parameters for our scheme were properly tuned to handle the noise, which degraded the performance of bootstrapping. To mitigate this issue we adopt the key unrolling technique introduced in [BMMP18, ZYL+18]. Though, we observe that ours require up to three times of overhead in bootstrapping time compared to the best-known results.

The higher parameters for our scheme also increases the size of key switching keys that are also used in bootstrapping. We propose a smooth converting method to reduce the size of key switching keys for our scheme, which could be of independent interest to other schemes as well.

We have reached the transmitted key size from the client to the server to be less than a megabyte (MB), which is more than $\times 10$ improvement compared to the best-known result. We implement all the proposed methods in the OpenFHE library [Ope22] and provide comparisons of our scheme with the previous results.

## 2 Preliminaries

All logarithms are base 2 unless otherwise indicated. For two vectors $\vec{a}$ and $\vec{b}$, we denote their inner product by $\langle \vec{a}, \vec{b} \rangle$. For a power of two $N$ we denote the $2N$-th cyclotomic ring by $\mathcal{R}_N := \mathbb{Z}[X]/\Phi_{2N}(X)$ and its quotient ring by $\mathcal{R}_{N,Q} := \mathcal{R}_N/Q\mathcal{R}_N$, where $\Phi_{2N}(X) = X^N + 1$. Ring elements in $\mathcal{R}_N$ are indicated in bold, e.g. $\boldsymbol{a} = \boldsymbol{a}(X)$. We denote the $L2$ and the infinity norm by $\|\cdot\|_2$ and $\|\cdot\|_\infty$ for a ring element $\boldsymbol{a} \in \mathcal{R}_N$ and a vector $\vec{a} \in \mathbb{Z}^n$. The norms of $\boldsymbol{a}$ and $\vec{a}$ are the norms of its coefficients and its elements, respectively.

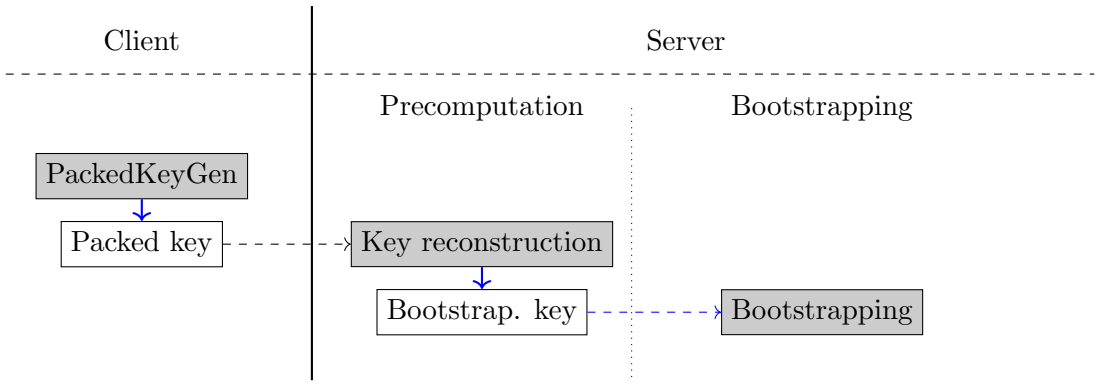


Figure 1: Summary of the proposed method

We write the floor, ceiling and round functions as $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$ and $\lfloor \cdot \rceil$, respectively. $\lfloor x \rceil_p$ denotes the nearest multiple of $p$ to $x$. For $q \in \mathbb{Z}$ and $q > 1$, we identify the ring $\mathbb{Z}_q$ with $[-q/2, q/2)$ as the representative interval, and for $x \in \mathbb{Z}$ we denote the centered remainder of $x$ modulo $q$ by $[x]_q \in \mathbb{Z}_q$. We extend these notations to elements of $\mathcal{R}_N$ by applying them coefficient-wise. We use $\boldsymbol{a} \leftarrow \mathsf{S}$ to denote uniform sampling from the set $\mathsf{S}$. We denote sampling according to a distribution $\chi$ by $\boldsymbol{a} \leftarrow \chi$.

## 2.1 Basic Lattice-based Encryption

We rewrite basic lattice-based encryptions following to [BGV14]. For positive integers $q$ and $n$, basic $\mathsf{LWE}$ encryption of $m \in \mathbb{Z}$ under the secret key $\vec{s} \leftarrow \chi_{\mathtt{key}}$ is defined as

$$\mathsf{LWE}_{n,q,\vec{s}}(m) = (b, \vec{a}) = (-\langle \vec{a}, \vec{s} \rangle + e + m, \vec{a}) \in \mathbb{Z}_q^{n+1},$$

where $\vec{a} \leftarrow \mathbb{Z}_q^n$ and error $e \leftarrow \chi_{\mathtt{err}}$. We will occasionally drop subscripts $n$, $q$, and $\vec{s}$ in $\mathsf{LWE}$ when they are clear from the context.

For a positive integer $Q$ and a power of two $N$, basic $\mathsf{RLWE}$ encryption of $\boldsymbol{m} \in \mathcal{R}$ under the secret key $\boldsymbol{z} \leftarrow \chi_{\mathtt{key}}$ is defined as

$$\mathsf{RLWE}_{N,Q,\boldsymbol{z}}(\boldsymbol{m}) := (\boldsymbol{b}, \boldsymbol{a}) = (-\boldsymbol{a} \cdot \boldsymbol{z} + \boldsymbol{e} + \boldsymbol{m}, \boldsymbol{a}) \in \mathcal{R}_{N,Q}^2,$$

where $\boldsymbol{a} \leftarrow \mathcal{R}_{N,Q}$ and $\boldsymbol{e} \leftarrow \chi_{\mathtt{err}}$. As in $\mathsf{LWE}$, we will occasionally drop subscripts $N$, $Q$, and $\boldsymbol{z}$ in $\mathsf{RLWE}$, when they are clear from the context.

## 2.2 Gadget Decomposition

A gadget toolkit over a modulus $Q$ consists of a fixed *gadget vector* $\vec{g} = (g_0, \ldots, g_{d-1})$ and *gadget decomposition* $h : \mathcal{R}_Q \to \mathcal{R}^d$, where $\langle h(\boldsymbol{t}), \vec{g} \rangle = \boldsymbol{t}$ and $h(\boldsymbol{t})$ is a *small* vector. In other words, $\|\boldsymbol{t}_i\|_\infty \leq B$ is satisfied for a boundary $B$, where $h(\boldsymbol{t}) = (\boldsymbol{t}_0, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_{d-1})$. In FHEW [DM15, MP21] the digit decomposition by factor $B/2$, such that $\vec{g} = (1, B, \ldots, B^{d-1})$ for $Q \leq B^d$ is used, and $h(\boldsymbol{t}) = (\boldsymbol{t}_0, \boldsymbol{t}_1, \ldots, \boldsymbol{t}_{d-1})$ is defined as the unique digit decomposition satisfying $\boldsymbol{t} = \sum_{i=0}^{d-1} \boldsymbol{t}_i B^i$ so that $\|\boldsymbol{t}_i\|_\infty \leq B/2$. We abuse the notation $h$ for *gadget decomposition* of an integer value as $h : \mathbb{Z}_q \to \mathbb{Z}^d$, where $\langle h(t), \vec{g} \rangle = t$ and $h(t)$ is a *small* vector.

We adapt the definitions of $\mathsf{RLWE}'$ and $\mathsf{RGSW}$ from [MP21]. For a gadget vector $\vec{g}$, we define $\mathsf{RLWE}'_{\boldsymbol{z}}(\boldsymbol{m})$ and $\mathsf{RGSW}_{\boldsymbol{z}}(\boldsymbol{m})$ as follows

$$\mathsf{RLWE}'_{\boldsymbol{z}}(\boldsymbol{m}) := (\mathsf{RLWE}\boldsymbol{z}(g_0 \cdot \boldsymbol{m}), \mathsf{RLWE}_{\boldsymbol{z}}(g_1 \cdot \boldsymbol{m}), \cdots, \mathsf{RLWE}_{\boldsymbol{z}}(g_{d-1} \cdot \boldsymbol{m})) \in \mathcal{R}_Q^{2\times d}$$
$$\mathsf{RGSW}_{\boldsymbol{z}}(\boldsymbol{m}) := \left(\mathsf{RLWE}'_{\boldsymbol{z}}(\boldsymbol{m}), \mathsf{RLWE}'_{\boldsymbol{z}}(\boldsymbol{z} \cdot \boldsymbol{m})\right) \in \mathcal{R}_Q^{2\times 2d}.$$

The scalar multiplication between an element $\boldsymbol{t} \in \mathcal{R}_Q$ and $\mathsf{RLWE}'(\boldsymbol{m})$ ciphertext

$$\odot : \mathcal{R}_Q \times \mathsf{RLWE}' \to \mathsf{RLWE}$$

is defined as

$$\begin{aligned}\boldsymbol{t} \odot \mathsf{RLWE}'(\boldsymbol{m}) &= \langle h(\boldsymbol{t}) = (\boldsymbol{t}_0, \cdots, \boldsymbol{t}_{d-1}), (\mathsf{RLWE}(g_0 \cdot \boldsymbol{m}), \cdots, \mathsf{RLWE}(g_{d-1} \cdot \boldsymbol{m}))\rangle \\ &= \sum_{i=0}^{d-1} \boldsymbol{t}_i \cdot \mathsf{RLWE}(g_i \cdot \boldsymbol{m}) = \mathsf{RLWE}\left(\sum_{i=0}^{d-1} g_i \cdot \boldsymbol{t}_i \cdot \boldsymbol{m}\right) = \mathsf{RLWE}(\boldsymbol{t} \cdot \boldsymbol{m}) \in \mathcal{R}_Q^2,\end{aligned}$$

For each error $\boldsymbol{e}_i$ in $\mathsf{RLWE}(g_i \cdot \boldsymbol{m})$, the error after multiplication is equal to $\sum_{i=0}^{d-1} \boldsymbol{t}_i \cdot \boldsymbol{e}_i$ which is small if $\boldsymbol{t}_i$ and $\boldsymbol{e}_i$ are small.

The multiplication between $\mathsf{RLWE}(\boldsymbol{m}) = (\boldsymbol{b}, \boldsymbol{a})$ and $\mathsf{RGSW}(\boldsymbol{m}')$ ciphertexts

$$\circledast : \mathsf{RLWE} \times \mathsf{RGSW} \to \mathsf{RLWE}$$

is defined as

$$(\boldsymbol{b}, \boldsymbol{a}) \circledast \mathsf{RGSW}(\boldsymbol{m}') = \boldsymbol{b} \odot \mathsf{RLWE}'(\boldsymbol{m}') + \boldsymbol{a} \odot \mathsf{RLWE}'(\boldsymbol{z} \cdot \boldsymbol{m}') = \mathsf{RLWE}(\boldsymbol{m} \cdot \boldsymbol{m}' + \boldsymbol{e} \cdot \boldsymbol{m}') \in \mathcal{R}_Q^2.$$

This result represents an $\mathsf{RLWE}$ encryption of the product $\boldsymbol{m} \cdot \boldsymbol{m}'$ with an additional error term $\boldsymbol{e} \cdot \boldsymbol{m}'$. In order to have $\mathsf{RLWE}(\boldsymbol{m}) \circledast \mathsf{RGSW}(\boldsymbol{m}') \approx \mathsf{RLWE}(\boldsymbol{m} \cdot \boldsymbol{m}')$, it is necessary to make the error term $\boldsymbol{e} \cdot \boldsymbol{m}'$ small. This can be achieved by using *small* $\boldsymbol{m}'$, e.g., monomial $\boldsymbol{m}' = \pm X^{\upsilon}$ as messages. The multiplication between $\mathsf{RLWE} \circledast \mathsf{RGSW}$ is naturally extended to $\mathsf{RLWE}'(\boldsymbol{m}) \circledast \mathsf{RGSW}(\boldsymbol{m}') \approx \mathsf{RLWE}'(\boldsymbol{m} \cdot \boldsymbol{m}')$ and $\mathsf{RGSW}(\boldsymbol{m}) \circledast \mathsf{RGSW}(\boldsymbol{m}') \approx \mathsf{RGSW}(\boldsymbol{m} \cdot \boldsymbol{m}')$, by performing the $\mathsf{RLWE} \circledast \mathsf{RGSW}$ multiplication for each $\mathsf{RLWE}$ component of left $\mathsf{RLWE}'$ or $\mathsf{RGSW}$.

## 2.3 Other Operations in LWE and RLWE

### Modulus Switching

The modulus switching operation converts $\mathsf{LWE}_Q(m)$ or $\mathsf{RLWE}_Q(\boldsymbol{m})$ in modulus $Q$ to $\mathsf{LWE}_{Q'}(\frac{Q'}{Q}m)$ or $\mathsf{RLWE}_{Q'}(\frac{Q'}{Q}\boldsymbol{m})$ in modulus $Q'$.

- For an input ciphertext $\mathsf{LWE}_{Q,\vec{z}}(m) = (b, a_1, \ldots, a_n) \in \mathbb{Z}_Q^{n+1}$, modulus switching to $Q'$ outputs

$$\mathsf{LWE}_{Q',\vec{z}}\left(\left\lfloor \frac{Q'}{Q} m \right\rceil\right) = \left(\left\lfloor \frac{Q'}{Q} b \right\rceil, \left\lfloor \frac{Q'}{Q} a_1 \right\rceil, \ldots, \left\lfloor \frac{Q'}{Q} a_n \right\rceil\right) \in \mathbb{Z}_{Q'}^{n+1}.$$

- For an input ciphertext $\mathsf{RLWE}_{Q,\boldsymbol{z}}(\boldsymbol{m}) = (\boldsymbol{b}, \boldsymbol{a}) \in \mathcal{R}_Q^2$, modulus switching to $Q'$ outputs

$$\mathsf{RLWE}_{Q',\boldsymbol{z}}\left(\left\lfloor \frac{Q'}{Q} \boldsymbol{m} \right\rceil\right) = \left(\left\lfloor \frac{Q'}{Q} \boldsymbol{b} \right\rceil, \left\lfloor \frac{Q'}{Q} \boldsymbol{a} \right\rceil\right) \in \mathcal{R}_{Q'}^2.$$

### Key Switching

The key switching operation converts $\mathsf{LWE}_{\vec{s}}(m)$ or $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m})$ encrypted by a secret key $\vec{s}$ or $\boldsymbol{s}$ to $\mathsf{LWE}_{\vec{z}}(m)$ or $\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m})$ encrypted by a new secret key $\vec{z}$ or $\boldsymbol{z}$ respectively. There are different variants of the key switching techniques and readers can refer to literature such as [KPZ21] for more details. We consider BV key switching type [BV11] as the most suitable for our purposes.

- For $\mathsf{LWE}$ the key switching generation and key switching are done as follows:
  - $\textsc{KeySwitchGen}(\vec{s}, \vec{z})$: Outputs $\mathtt{ksk} = \left\{\mathsf{LWE}_{\vec{z}}(kB^i s_j)\right\}_{i\in[0,d-1], j\in[0,n-1], k\in(-B/2,B/2]}$
  - $\textsc{KeySwitch}_{\vec{s}\to\vec{z}}(\mathtt{ksk}, \mathsf{LWE}_{\vec{s}}(m))$: Given $\mathsf{LWE}_{\vec{s}}(m) = (b, \vec{a})$, it evaluates
  $$\mathsf{LWE}_{\vec{z}}(m) = (b, 0, \ldots, 0) + \sum \mathsf{LWE}(h(a_j)_i \cdot B^i s_j) \pmod{Q}$$
  where $h(a_j)_i$ is the $i$-th element of $h(a_j)$. The key switching error is equal to the sum of $dn$ of $\mathsf{LWE}$ errors.
- For $\mathsf{RLWE}$ the key switching generation and key switching are done as follows:
  - $\textsc{KeySwitchGen}(\boldsymbol{s}, \boldsymbol{z})$ for $\mathsf{RLWE}$: Outputs $\mathtt{ksk} = \mathsf{RLWE}'_{\boldsymbol{z}}(\boldsymbol{s})$.
  - $\textsc{KeySwitch}_{\boldsymbol{s}\to\boldsymbol{z}}(\mathtt{ksk}, \mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}))$: Given $\mathsf{RLWE}_{\boldsymbol{s}}(\boldsymbol{m}) = (\boldsymbol{b}, \boldsymbol{a})$, it evaluates
  $$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}) = (\boldsymbol{b}, 0) + \boldsymbol{a} \odot \mathsf{RLWE}'_{\boldsymbol{z}}(\boldsymbol{s}) \pmod{Q}.$$
  The key switching error is equal to the error of a $\mathcal{R} \odot \mathsf{RLWE}'$ multiplication.

**Remark 1.** *Note that whereas the error is smaller for $\mathsf{LWE}$ key switching, compared to $\mathsf{RLWE}$ key switching with similar parameters, the key size is much bigger. This trade-off will be useful in our scheme design.*

### Automorphisms of RLWE

There are $N$ automorphisms of $\mathcal{R}$, namely $\psi_t : \mathcal{R} \to \mathcal{R}$ given by $\boldsymbol{a}(X) \mapsto \boldsymbol{a}(X^t)$ for $t \in \mathbb{Z}^*_{2N}$. Automorphism over $\mathsf{RLWE}$ instances can be defined as follows.

- $\textsc{EvalAuto}(\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}), \mathtt{ak}_t)$: Given $\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}(X)) = (\boldsymbol{b}(X), \boldsymbol{a}(X))$, it applies $\psi_t$ to both $\boldsymbol{b}(X)$ and $\boldsymbol{a}(X)$ and obtains $(\boldsymbol{b}(X^t), \boldsymbol{a}(X^t))$ which is an $\mathsf{RLWE}$ encryption of $\boldsymbol{m}(X^t)$ under the secret key $\boldsymbol{z}(X^t)$. Then it performs key switching from $\boldsymbol{z}(X^t)$ to $\boldsymbol{z}$ using the automorphism key $\mathtt{ak}_t = \mathsf{RLWE}'_{\boldsymbol{z}}(\boldsymbol{z}(X^t))$ and outputs $\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}(X^t)) = \mathsf{RLWE}_{\boldsymbol{z}}(\psi_t(\boldsymbol{m}))$.

The additional error after applying an automorphism is equal to the key switching error as an automorphism $\psi_t$ is a norm-preserving map.

## 2.4 FHEW-like Bootstrapping

FHEW-like bootstrapping can be deployed for any other binary operation [DM15, CGGI20] and we focus on $\mathsf{NAND}$ operation for simplicity in this paper. We refer readers to [CGGI20] for torus arithmetics and to [BIP+22] for schemes that rely on the NTRU problem.

**Blind rotation**

At a high level, the bootstrapping consists of blind rotation and key & modulus switching. Given an $\mathsf{LWE}$ ciphertext $(b, \vec{a})$ encrypted under the secret key $\vec{s}$, the goal of blind rotation is to find

$$\mathsf{RLWE}(\mathtt{f} \cdot X^{b+\vec{a}\cdot\vec{s}}),$$

where the $\mathsf{LWE}$ ciphertext modulus $q = 2N$ and $\mathtt{f}$ is a look-up table polynomial corresponding to the operation [DM15, CGGI20][1]. The basic building block of blind rotation is accumulation. Several techniques have been presented for accumulation that supports different secret key distributions with different performance and key size tradeoffs [DM15, CGGI20, BMMP18, ZYL+18, KDE+21, BIP+22, LMK+23]. In this paper, we mostly focus on CGGI [CGGI20] technique and its variants BMMP [BMMP18] and ZYLZD [ZYL+18], which are all efficient for the case of binary secret key distribution $\vec{s}$.

- **CGGI Variant**: Let $\mathtt{g}$ be a polynomial and $\mathsf{RGSW}(s)$ be the blind rotation key. The blind rotation of the CGGI variant is based on the following accumulation:

  $$\mathsf{RLWE}(\mathtt{g}) \circledast (\mathbf{1} + (X^a - 1) \cdot \mathsf{RGSW}(s)) = \mathsf{RLWE}(\mathtt{g} \cdot X^{as}).$$

  which is sequentially applied $n$ times to $\mathsf{RLWE}(\mathtt{f})$.

A key unrolling technique is proposed to improve the performance of blind rotation by reducing the number of $\circledast$ operations which is the most costly operation in accumulation [ZYL+18, BMMP18]. The technique of the BMMP variant is slightly better in terms of key size and slightly worse in terms of error growth than that of the ZYLZD variant. The basic idea is to group elements of the secret key $\vec{s}$ and $\vec{a}$ of the $\mathsf{LWE}$ ciphertext and accumulate them together.

- **BMMP Variant**: Let $u$ be an unrolling factor and $\vec{v} \in V = \{0,1\}^u \setminus \vec{0}$. Let $\vec{s}_\ell = (s_\ell, s_{\ell+1}, \ldots, s_{\ell+u-1})$ denote the group of $\vec{s}$. For $\ell = 0, 1, \ldots, n/u - 1$, the key values are defined as $\delta_{\vec{v},\ell} = 1$ if $\vec{v} = \vec{s}_\ell$, and 0 otherwise. The blind rotation keys are generated by encrypting $\delta_{\vec{v},\ell}$ i.e. $\mathsf{RGSW}(\delta_{\vec{v},\ell})$.

  Let $\mathtt{g}$ be a polynomial and $\delta_{\vec{v},\ell}$ and $\vec{v}$ denote the key values defined above. The blind rotation of the BMMP variant with key unrolling is based on the following accumulation:

  $$\mathsf{RLWE}(\mathtt{g}) \circledast \left(\mathbf{1} + \sum_{\vec{v}\in V} \left((X^{\sum_{j=0}^{u-1} v_j a_j} - 1) \cdot \mathsf{RGSW}(\delta_{\vec{v},\ell})\right)\right) = \mathsf{RLWE}(\mathtt{g} \cdot X^{\sum_{j=0}^{u-1} a_j s_j})$$

  which sequentially applied $n/u$ times to $\mathsf{RLWE}(\mathtt{f})$.

- **ZYLZD Variant**: The difference between the ZYLZD variant from the BMMP variant is that here we assume $\vec{v} \in V = \{0,1\}^u$. The blind rotation keys are generated as $\mathsf{RGSW}(\delta_{\vec{v},\ell})$. The blind rotation of the ZYLZD variant with key unrolling is based on the following accumulation:

  $$\mathsf{RLWE}(\mathtt{g}) \circledast \left(\sum_{\vec{v}\in V} \left(X^{\sum_{j=0}^{u-1} v_j a_j} \cdot \mathsf{RGSW}(\delta_{\vec{v},\ell})\right)\right) = \mathsf{RLWE}(\mathtt{g} \cdot X^{\sum_{j=0}^{u-1} a_j s_j})$$

  which sequentially applied $n/u$ times to $\mathsf{RLWE}(\mathtt{f})$.

[1] The ciphertext modulus can be any other $q$ that divides $2N$, but for brevity we only consider $q = 2N$ in this paper. We refer the readers to [LMK+23] for a generalized definition of blind rotation.

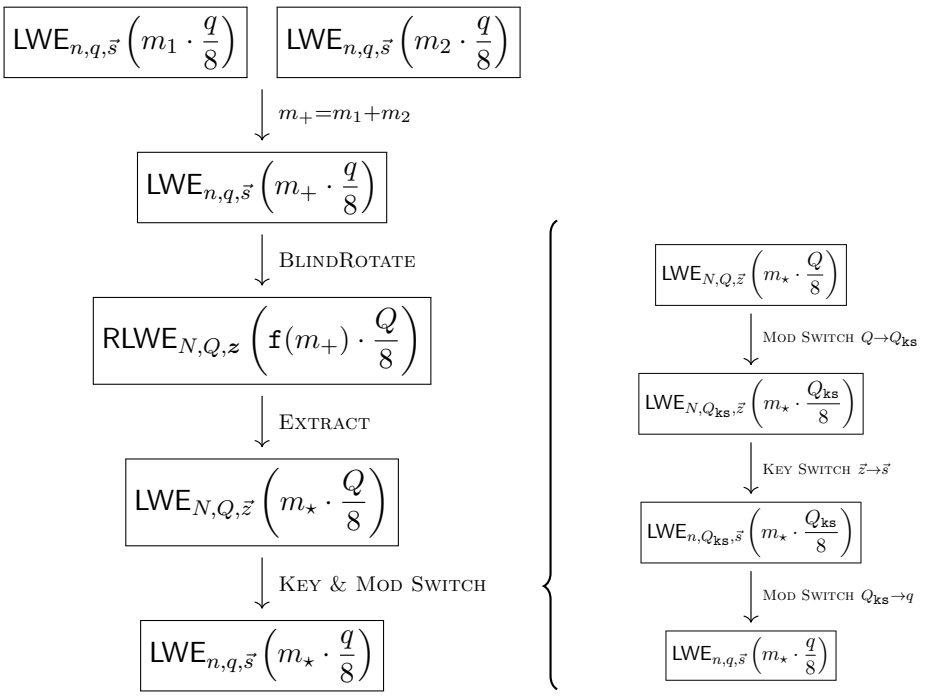


Figure 2: FHEW-like bootstrapping for NAND gate [MP21]

**Remark 2.** *By increasing $u$ the number of addition between RGSW ciphertexts increases by a factor of $2^u$, so increasing $u$ does not always improve performance. Also, errors are accumulated during addition which could increase the parameters to maintain the security level and FHEW-like schemes are very sensitive to the size of the parameters.*

The full bootstrapping procedure for FHEW-like schemes for ring arithmetics is described in Figure 2.

# 3 Light-Key Fully Homomorphic Encryption

There are two types of keys involved in FHEW-like bootstrapping: key-switching keys and blind rotation keys. As we mentioned above, the size of blind rotation keys is quite huge and it causes higher communication costs. In this section, we present a method for packing the keys into a small amount of RLWE ciphertexts on the client side and reconstructing them to blind rotation keys on the server side. This approach reduces the size of blind rotation keys the client sends, however, the key reconstruction introduced additional noise so that the parameters should be higher, which causes increasing bootstrapping runtime. To deal with such a problem, we adopt key unrolling methods and an approximate gadget decomposition method. Besides, since the higher parameters increase the size of key switching keys, we propose a smooth converting process to reduce the total size of key switching keys. Our approach offers trade-offs between communication costs depending on the size of keys to be transferred and computational costs performed on the server side.

### 3.1 Blind Rotation Key Packing and Reconstruction

Recall that blind rotation keys $\left\{\mathsf{RGSW}_{\boldsymbol{z}}(s_i) = (\mathsf{RLWE}'_{\boldsymbol{z}}(s_i)), \mathsf{RLWE}'_{\boldsymbol{z}}(s_i \cdot \boldsymbol{z})\right\}$ are $\mathsf{RGSW}$ encryptions of $s_i \in \{0,1\}$ which are corresponding to elements of the binary LWE secret key $\vec{s}$. We observe the first parts of $\mathsf{RGSW}_{\boldsymbol{z}}(s_i)$ consisting of a lot of $\mathsf{RLWE}$ encryptions for each $s_i$ i.e. $\mathsf{RLWE}(B^j s_i)$ where $(1, B, B^2, \ldots, B^{d_g-1})$ is a gadget vector, which causes the huge size of keys. So instead of generating a separate $\mathsf{RLWE}$ encryption for each $B^j s_i$, we pack them into coefficients of a small number of polynomials $\boldsymbol{\alpha}_k(X)$ and generate $\mathsf{RLWE}(\boldsymbol{\alpha}_k(X))$. For example, the first packed polynomial can be a form of

$$\boldsymbol{\alpha}_0(X) = s_0 + Bs_0X + B^2s_0X^2 + \cdots + B^{d_g-1}s_0X^{d_g-1} + s_1X^{d_g} + \cdots + B^j s_i X^{N-1}$$

where $i = N/d_g$ and $j \equiv N \pmod{d_g}$. With this order, the values $B^j s_i$ for any integer $i \in [0, \ldots, n-1]$ and $j \in [0, \ldots, d_g - 1]$ are placed into $t$-th coefficient in $\alpha_k(X)$ polynomial where $k = \left\lfloor \frac{i \cdot d_g + j}{N} \right\rfloor$ and $t = i \cdot d_g + j \pmod{N}$. After that all $\alpha_k(X)$ polynomials are encrypted under the secret key $\boldsymbol{z}$ and the second parts of $\mathsf{RGSW}_{\boldsymbol{z}}(s_i)$ will be reconstructed from the first parts so that the number of $\mathsf{RLWE}$ ciphertexts to be transferred will be reduced from $2d_gn$ to $\left\lceil \frac{d_g n}{N} \right\rceil$. Note that the order of coefficients to pack is not constrained.

To extract $\mathsf{RLWE}(B^j s_i)$ from the packed ciphertext $\mathsf{RLWE}(\boldsymbol{\alpha}_k(X))$ during unpacking, we use the following property of the $\mathsf{RLWE}$ automorphism operation. Simply, given a polynomial $\boldsymbol{m} = m_0 + m_1X + m_2X^2 + \cdots + m_{N-1}X^{N-1}$, we have $\psi_{N+1}(\boldsymbol{m}) = m_0 - m_1X + m_2X^2 - \cdots - m_{N-1}X^{N-1}$, which means this operation inverses the sign of each odd coefficient and the odd or even coefficients of the polynomial $\boldsymbol{m} \pm \psi_{N+1}(\boldsymbol{m})$ will be zero. With $\log N$ automorphisms by $N/2+1, N/4+1, \ldots, 2$, we can take a single non-zero coefficient from the polynomial $\boldsymbol{m}$. This property is used in Algorithm 3 which demonstrates how to homomorphically extract each coefficient of $\boldsymbol{\alpha}_k(X)$ and obtain $N$ ciphertexts with only one non-zero coefficient, which are encryptions of $B^j s_i$. Note that this algorithm returns encryptions of $NB^j s_i$, so the coefficients of $\boldsymbol{\alpha}_k(X)$ should be multiplied by $N^{-1} \pmod{Q}$ before encryption to cancel out $N$ during the extraction. The corresponding key packing algorithm is described in Algorithm 1. Meanwhile, Algorithm 3 requires automorphism keys $\mathtt{ak}_{2^w+1}$ for $w = 1, \ldots, \log N$, so they also have to be generated on the client side as a part of packed blind rotation keys.

**Algorithm 1** Blind Rotation Key Packing

**procedure** PACKBRKEY($\vec{s}$, $\boldsymbol{z}$)
  **for** $(i = 0; i < n; i = i + 1)$ **do**
    **for** $(j = 0; j < d; j = j + 1)$ **do**
      Put $[N^{-1}]_Q \cdot B^j s_i$ into the coefficient of a polynomial $\boldsymbol{\alpha}_k(X)$
      **if** ($\boldsymbol{\alpha}_k(X)$ is fully packed) **then**
        Move to the next polynomial $\boldsymbol{\alpha}_{k+1}(X)$
  **for** $(k = 0; k < \left\lceil \frac{nd}{N} \right\rceil; k = k + 1)$ **do**
    $\mathtt{pbk}_k \leftarrow \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{\alpha}_k(X))$
**return** $\{\mathtt{pbk}_k\}$

After obtaining $\mathsf{RLWE}(B^j s_i)$, the second parts of the blind rotation keys $\mathsf{RLWE}(B^j s_i \cdot \boldsymbol{z})$ can be recontructed from $\mathsf{RLWE}(B^j s_i) = (\boldsymbol{b}_{i,j}, \boldsymbol{a}_{i,j})$ through the following equation which we call EVALSQUAREMULT with an additional square key $\mathtt{sqk} = \mathsf{RLWE}'_{\boldsymbol{z}}(\boldsymbol{z}^2)$:

$$\boldsymbol{b}_{i,j} \cdot (0,1) + \boldsymbol{a}_{i,j} \odot \mathtt{sqk} = \mathsf{RLWE}(\boldsymbol{b}_{i,j} \cdot \boldsymbol{z} + \boldsymbol{a}_{i,j} \cdot \boldsymbol{z}^2) = \mathsf{RLWE}(B^j s_i \cdot \boldsymbol{z}). \tag{1}$$

The square key $\mathtt{sqk}$ should also be generated on the client side as a part of packed blind rotation keys. The full key packing algorithm performed on the client side is given in Algorithm 2 and the number of keys becomes asymptotically $O(\log N + n/N) = O(\log N)$ as $n/N < 1$.

**Algorithm 2** Packed Key Generation

**procedure** PACKEDKEYGEN($\vec{s}, \boldsymbol{z}$)
  $\{\mathtt{pbk}_k\} \leftarrow$ PACKBRKEY($\vec{s}, \boldsymbol{z}$), for $k \in \left\{0, 1, 2, \ldots, \left\lceil \frac{d_g n}{N} \right\rceil - 1\right\}$
  **for** $(t = 2; t \leq N; t = 2 \cdot t)$ **do**
    $\mathtt{ak}_t \leftarrow$ KEYSWITCHGEN($\boldsymbol{z}(X^{t+1}), \boldsymbol{z}$)
  $\mathtt{sqk} \leftarrow \mathsf{RLWE}'_{\boldsymbol{z}}(\boldsymbol{z}^2)$
**return** $\{\{\mathtt{pbk}_k\}, \{\mathtt{ak}_t\}, \mathtt{sqk}\}$

Given a pack of keys for blind rotation, the original blind rotation keys $\mathsf{RGSW}_{\boldsymbol{z}}(s_i)$ should be reconstructed on the server side. First, it applies Algorithm 3 to $\mathtt{pbk}_k$ and obtains $\mathsf{RLWE}_{\boldsymbol{z}}(B^j s_i)$ which are the first parts of $\mathsf{RGSW}_{\boldsymbol{z}}(s_i)$. To reconstruct the second parts $\mathsf{RLWE}_{\boldsymbol{z}}(B^j s_i \cdot \boldsymbol{z})$ of $\mathsf{RGSW}_{\boldsymbol{z}}(s_i)$, it performs EVALSQUAREMULT to $\mathsf{RLWE}'_{\boldsymbol{z}}(B^j s_i)$ with $\mathtt{sqk}$. Finally, it rearranges $\mathsf{RLWE}'_{\boldsymbol{z}}(B^j s_i)$ and $\mathsf{RLWE}'_{\boldsymbol{z}}(B^j s_i \cdot \boldsymbol{z})$ to return $\mathsf{RGSW}_{\boldsymbol{z}}(s_i)$. The full reconstruction algorithm on the server side is given in Algorithm 4.

**Algorithm 3** Coefficients Extraction

**procedure** EXTRACTCOEFFS($\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{\alpha}(X)), \{\mathtt{ak}_{2^w+1}\}$)
  $\boldsymbol{\alpha}(X) = \alpha_0 + \alpha_1 X + \alpha_2 X^2 + \cdots + \alpha_{N-1} X^{N-1}$
  $\mathtt{res}_0 \leftarrow \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{\alpha}(X))$
  **for** $(\ell = N; \ell > 1; \ell = \ell/2)$ **do**
    **for** $(j = 0; j < N/\ell; j = j + 1)$ **do**
      $\mathtt{old} \leftarrow \mathtt{res}_j$
      $\mathtt{tmp} \leftarrow$ EVALAUTO($\mathtt{old}, \mathtt{ak}_{\ell+1}$)
      $\mathtt{res}_j = \mathtt{old} + \mathtt{tmp}$
      $\mathtt{res}_{j+N/\ell} = X^{-N/\ell} \cdot (\mathtt{old} - \mathtt{tmp})$
**return** $\{\mathtt{res}_i\} = \{\mathsf{RLWE}_{\boldsymbol{z}}(N\alpha_i)\}$ for $i \in \{0, 1, \ldots, N-1\}$

**Algorithm 4** Key Reconstruction

**procedure** KEYRECONSTRUCT($\{\{\mathtt{pbk}_k\}, \{\mathtt{ak}_t\}, \mathtt{sqk}\}$)
  **for** $(k = 0; k < \left\lceil \frac{d_g n}{N} \right\rceil; k = k + 1)$ **do**
    Find $\mathsf{RLWE}_{\boldsymbol{z}}(B^j s_i)$ using EXTRACTCOEFFS($\mathtt{pbk}_k, \{\mathtt{ak}_t\}$)
    $\mathsf{RLWE}_{\boldsymbol{z}}(B^j s_i \boldsymbol{z}) \leftarrow$ EVALSQUAREMULT($\mathsf{RLWE}(B^j s_i), \mathtt{sqk}$)
    Rearrange them into $\mathsf{RGSW}_{\boldsymbol{z}}(s_i)$
**return** $\{\mathsf{RGSW}_{\boldsymbol{z}}(s_i)\}$

Table 1: Trade-offs by the unrolling factor $u$.

| Scheme | Key size (# of RGSW) | # of NTTs |
|---|---|---|
| CGGI/BMMP/ZYLZD | $O(n \cdot 2^u)$ | $O(1/u)$ |
| Proposed | $O(\log N + 2^u)$ | $O(1/u)$ |

**Remark 3.** *For ternary secret keys, the blind rotation keys are* $\left\{\mathsf{RGSW}_{\boldsymbol{z}}(s_i^+), \mathsf{RGSW}_{\boldsymbol{z}}(s_i^-)\right\}$*, where* $s_i^+, s_i^- \in \{0,1\}$ *are corresponding to whether the elements of the LWE secret key* $\vec{s}$ *are* $1$ *or* $-1$*, respectively [KDE+21, BIP+22]. In this case, we need to pack all those values* $B^j s_i^+$ *and* $B^j s_i^-$ *into coefficients of polynomials* $\alpha_k(X)$ *and extract them properly.*

### 3.1.1 Key Unrolling Optimization

The proposed key packing method is described based on the blind rotation key structure of the CGGI scheme. We can also apply this method to other schemes. As described in 2.4, key unrolling techniques are introduced to improve the performance of blind rotation in [BMMP18, ZYL+18]. However, they increase the size of blind rotation keys and our key packing method would be helpful to reduce the size of keys generated on the client side. More precisely, in comparison with CGGI, the key size is increased by $2^u/u$ times and the number of NTT is decreased by $1/u$ where $u$ is the unrolling factor. That is, we need $O(n)$ RGSW keys for blind rotation in [CGGI20], and the number of keys becomes asymptotically $O(n \cdot 2^u/u) = O(n \cdot 2^u)$ with key unrolling. Such increased key size can be reduced to $O(\log N + n/N \cdot 2^u/u) = O(\log N + 2^u)$ by applying our key packing method. The trade-offs between the key size and the number of NTT are represented in Table 1. It is noted that Table 1 is not a fair comparison of [CGGI20] and the proposed method, as parameters of FHEW-like schemes are quite an error sensitive and higher parameters are required for key reconstruction. The table just indicates that key unrolling is better suited for the proposed key packing scenario because it mitigates the disadvantages of large key sizes.

## 3.2 Smooth Converting Process

The proposed key packing method requires higher parameters due to the additional noise introduced during key reconstruction and it increases the size of key switching keys. To deal with this problem, we modify the bootstrapping procedure.

In the original FHEW bootstrapping, all steps following blind rotation can be considered as substeps of converting an $\mathsf{RLWE}_{N,Q,\boldsymbol{z}}(m \cdot \frac{Q}{8})$ ciphertext into an $\mathsf{LWE}_{n,q,\vec{s}}(m \cdot \frac{q}{8})$ ciphertext, where $n < N$ and $q < Q$. For example, in [MP21] the converting process consists of extracting $N$ LWE ciphertexts from the RLWE ciphertext, followed by modulus switching from $Q$ to the intermediate modulus $Q_{\mathtt{ks}}$ and key switching from $\vec{z}$ to $\vec{s}$, and finally modulus switching from $Q_{\mathtt{ks}}$ to $q$, where $q < Q_{\mathtt{ks}} < Q$. Modulus switching to the intermediate modulus $Q_{\mathtt{ks}}$ is introduced to ensure that key switching keys satisfy the required security level. The full flow is given in Equation 2.

$$\mathsf{RLWE}_{N,Q} \xrightarrow{Ext} \mathsf{LWE}_{N,Q} \xrightarrow{MS} \mathsf{LWE}_{N,Q_{\mathtt{ks}}} \xrightarrow{KS} \mathsf{LWE}_{n,Q_{\mathtt{ks}}} \xrightarrow{MS} \mathsf{LWE}_{n,q} \tag{2}$$

Now we propose the improved process for FHEW bootstrapping that we call a smooth converting process. First, if key switching is performed in RLWE form in Equation 2, when $n$ is a power of two as discussed in [DM15], it will decrease the size of key switching keys that are transmitted as the size of LWE key switching keys is bigger than that of RLWE key switching keys. To do that, the

converting steps should be modified such that modulus switching and key switching for an RLWE ciphertext followed by extraction of LWE ciphertexts (see Equation 3).

$$\mathsf{RLWE}_{N,Q} \xrightarrow{MS} \mathsf{RLWE}_{N,Q_{\mathtt{ks}}} \xrightarrow{KS} \mathsf{RLWE}_{n,Q_{\mathtt{ks}}} \xrightarrow{Ext} \mathsf{LWE}_{n,Q_{\mathtt{ks}}} \xrightarrow{MS} \mathsf{LWE}_{n,q} \tag{3}$$

However, RLWE key switching produces more noise than LWE key switching and RLWE key switching in small parameters is very sensitive to noise. So for our solution, we get the best results from both RLWE and LWE key switchings by splitting the procedures into two steps with an extra parameter $Q_{\mathtt{sm}}$ as follows:

$$\mathsf{RLWE}_{N,Q} \xrightarrow{MS} \mathsf{RLWE}_{N,Q_{\mathtt{sm}}} \xrightarrow{KS} \mathsf{RLWE}_{N_{\mathtt{sm}},Q_{\mathtt{sm}}} \xrightarrow{Ext} \mathsf{LWE}_{N_{\mathtt{sm}},Q_{\mathtt{sm}}} \xrightarrow{MS} \mathsf{LWE}_{N_{\mathtt{sm}},Q_{\mathtt{ks}}} \xrightarrow{KS} \mathsf{LWE}_{n,Q_{\mathtt{ks}}} \xrightarrow{MS} \mathsf{LWE}_{n,q} \tag{4}$$

where $q < Q_{\mathtt{ks}} < Q_{\mathtt{sm}} < Q$. The additional noise from key switching is accumulated in the LSB of the ciphertext. The first key switching is less sensitive to noise due to the higher modulus $Q_{\mathtt{sm}}$, so it can be done in RLWE form with the small size of keys. The latter key switching is performed in LWE form with the smaller modulus $Q_{\mathtt{ks}}$.

**Key size and noise growth with smooth converting** We compare the key size and noise growth of previous converting (Equation (2)) and the smooth converting (Equation (4)). The LWE key switching key can be greatly reduced by using smooth converting, and show that smooth converting provides less key size and noise growth compared to the previous one. In (2), we only need LWE key switching key

$$\left\{\mathsf{LWE}_{Q_{\mathtt{ks}},\vec{z}}(kB^i s_j)\right\}_{i\in[0,d-1],j\in[0,n-1],k\in(-B/2,B/2]}.$$

We normally use small $d_{\mathtt{ks}}$ to minimize the noise growth as the LWE key switching error takes a significant portion of the bootstrapping error. Hence, the key size is

$$d_{\mathtt{ks}} Q_{\mathtt{ks}}^{1/d_{\mathtt{ks}}} N(n+1)\log Q_{\mathtt{ks}}.$$

In (4), we need need additional RLWE$'$ keys for the first key switching

$$\mathsf{RLWE}'_{N_{\mathtt{sm}},Q_{\mathtt{sm}},z_{sm}}(\boldsymbol{z}_0),\ldots,\mathsf{RLWE}'_{N_{\mathtt{sm}},Q_{\mathtt{sm}},z_{sm}}\left(\boldsymbol{z}_{\frac{N}{N_{\mathtt{sm}}}-1}\right),$$

where

$$\boldsymbol{z}_j = \sum_{i=0}^{N_{\mathtt{sm}}-1} z_{\frac{N}{N_{\mathtt{sm}}}\cdot i+j}\cdot X^i \in \mathbb{Z}[X]/\langle X^{N_{\mathtt{sm}}}+1\rangle.$$

For example, when $N_{\mathtt{sm}} = N/2$, $\boldsymbol{z}_0$ and $\boldsymbol{z}_1$ are the polynomials of degree $N_{\mathtt{sm}}-1$ with the even and odd coefficients of $\boldsymbol{z}$, respectively. The size of these RLWE$'$ keys are $N/N_{\mathtt{sm}}\cdot 2d_{\mathtt{sm}}N_{\mathtt{sm}}\log Q_{\mathtt{sm}}$. We also need LWE key switching key for the second key switching, whose key size is $d_{\mathtt{ks}}Q_{\mathtt{ks}}^{1/d_{\mathtt{ks}}}N_{\mathtt{sm}}(n+1)\log Q_{\mathtt{ks}}$. Thus, the total key size is given as

$$2d_{\mathtt{sm}}N\log Q_{\mathtt{sm}} + d_{\mathtt{ks}}Q_{\mathtt{ks}}^{1/d_{\mathtt{ks}}}N_{\mathtt{sm}}(n+1)\log Q_{\mathtt{ks}}.$$

As $d_{\mathtt{sm}}$ is small, we can see that the LWE key switching key size dominates. With smooth converting, the key size is reduced by almost a factor $N/N_{\mathtt{sm}}$, compare to not smooth variant.

The noise variance of ciphertext right after the LWE key switching with (2) is

$$2N\sigma_{\mathtt{err}}^2,$$

while the noise variance after the LWE key switching with (4) is

$$\frac{Q_{\mathtt{ks}}^2}{Q_{\mathtt{sm}}^2}\left(d_{\mathtt{sm}}N\frac{Q_{\mathtt{sm}}^{2/d_{\mathtt{sm}}}}{12}\sigma_{\mathtt{err}}^2\right)+2N_{\mathtt{sm}}\sigma_{\mathtt{err}}^2.$$

The noise from LWE key switching again dominates, given that $Q_{\mathtt{ks}} \ll Q_{\mathtt{sm}}$. With smooth converting, the noise variance is reduced by almost a factor $N/N_{\mathtt{sm}}$, compare to not smooth variant.

The technique of smooth converting is not exclusive to the proposed scheme and can be effectively utilized in various homomorphic encryption approaches, including FHEW-like schemes and scheme switching. By reducing the key size and noise growth, this technique offers independent interest as it directly impacts the performance of blind rotation in FHEW-like schemes, where noise growth is directly proportional to the parameter size.

#### 3.2.1 Scheme Steps Modification

In the original FHEW bootstrapping, input and output ciphertexts are in the smallest parameters $\mathsf{LWE}_{n,q}$. While keeping the ciphertexts in the small parameters could be beneficial for transmission, this method is not efficient in terms of error growth. In [CGGI20, LMK+23], a modified scheme was proposed, where the input and output ciphertexts are in higher parameters $\mathsf{LWE}_{N,Q}$, and converting from $\mathsf{LWE}_{N,Q}$ to $\mathsf{LWE}_{n,q}$ is processed right before the blind rotation step (see Figure 3). This modification decreases the total error and thus helps to use smaller parameters for the same failure probability threshold.

We can also apply a similar modification that has input and output ciphertexts to be in the higher parameters $\mathsf{LWE}_{N_{\mathtt{sm}},Q_{\mathtt{sm}}}$. The converting steps are then split into two parts. First, after performing an operation to input ciphertexts in $\mathsf{LWE}_{N_{\mathtt{sm}},Q_{\mathtt{sm}}}$ we convert it into $\mathsf{LWE}_{n,q}$, and after the blind rotation step we convert it from $\mathsf{RLWE}_{N,Q}$ to $\mathsf{LWE}_{N_{\mathtt{sm}},Q_{\mathtt{sm}}}$ with the smooth converting process. The full procedure is shown in Figure 4.

### 3.3 Approximate Gadget Decomposition

An approximate gadget decomposition was first proposed in [CGGI20] to define TGSW, which is the GSW construction over the real torus, and its external product with TLWE. Its simple variant working on the integer was introduced in [LMK+23] to reduce the key size and runtime for the external product between RLWE and RGSW. In their construction, the keys are considered as fresh RGSW ciphertexts which have small errors, thus $|e| < B$, and the first part of the decomposed value (which will be multiplied by 1 in a gadget vector) can be ignored while performing the external product. However, the reconstructed RGSW key in our technique has a much larger error than a fresh ciphertext and the error can be greater than $B$. Thus, extending the gadget decomposition in [LMK+23] is not suitable for our construction. Instead, we need to redefine an approximate gadget decomposition on the integer to be similar to the definition in [CGGI20].

We introduce an approximation factor $\delta$ such that $B^d\delta < Q$. The approximate gadget vector is given as $\vec{g}_\delta = (\delta, B\delta, \ldots, B^{d-1}\delta)$, and the approximate decomposition of $\boldsymbol{a} \in \mathcal{R}_Q$ is to find $\boldsymbol{a}_i$ that

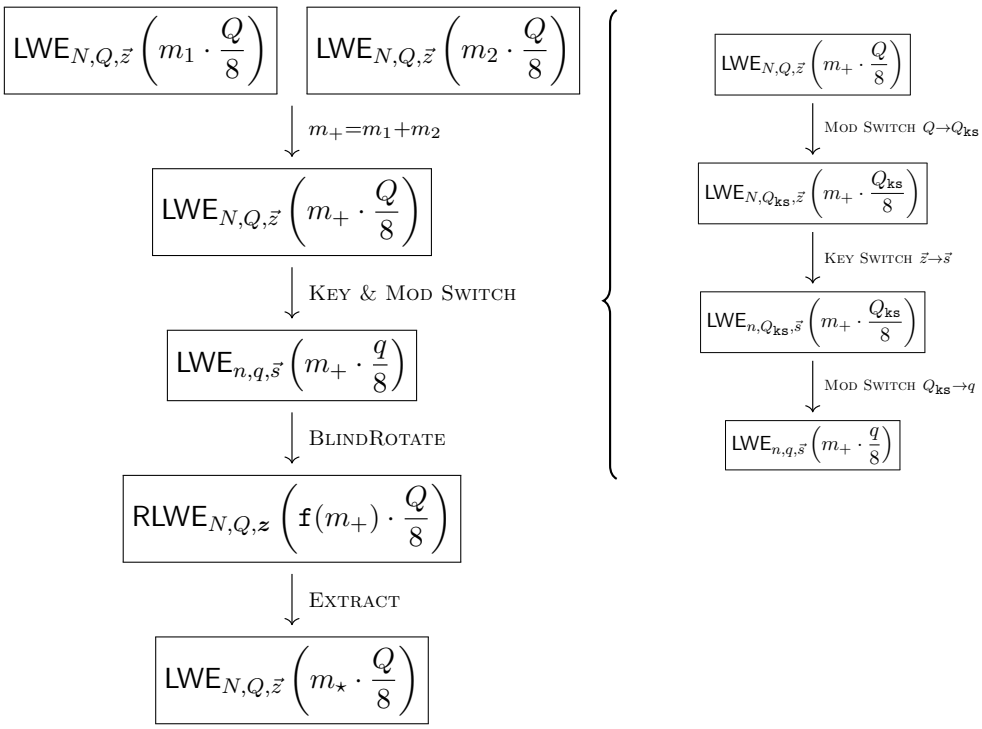


Figure 3: Steps modification bootstrapping for NAND gate [CGGI20, LMK+23]

minimizes decomposition error $\left\|\boldsymbol{a} - \sum_{j=0}^{d-1} \boldsymbol{a}_i \cdot \delta B^j\right\|_\infty$, where $\|\boldsymbol{a}_i\|_\infty \leq B/2$. Also, $\mathsf{RLWE}'(\boldsymbol{m}) \in \mathcal{R}_Q^{2\times d}$ is defined as follows

$$\mathsf{RLWE}'(\boldsymbol{m}) = \Big(\mathsf{RLWE}(\delta \cdot \boldsymbol{m}), \mathsf{RLWE}(B\delta \cdot \boldsymbol{m}), \ldots, \mathsf{RLWE}(B^{d-1}\delta \cdot \boldsymbol{m})\Big).$$

The external product $\boldsymbol{a} \odot \mathsf{RLWE}'(\boldsymbol{m})$ is defined as

$$\begin{aligned}\langle(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{d-1}), \Big(\mathsf{RLWE}(\delta \cdot \boldsymbol{m}), \mathsf{RLWE}(B\delta \cdot \boldsymbol{m}), \ldots, \mathsf{RLWE}(B^{d-1}\delta \cdot \boldsymbol{m})\Big)\rangle \\ = \sum_{j=0}^{d-1} \mathsf{RLWE}(\boldsymbol{a}_j \cdot B^j\delta \cdot \boldsymbol{m}) \approx \mathsf{RLWE}(\boldsymbol{a} \cdot \boldsymbol{m}).\end{aligned}$$

Following to [CGGI20], we use the notation $\mathsf{Err}(\boldsymbol{c})$ and $\mathsf{Var}(\mathsf{Err}(\boldsymbol{c}))$ which denote the error of $\boldsymbol{c}$ and the variance of $\mathsf{Err}(\boldsymbol{c})$, respectively.

**Proposition 1.** *Let $\sigma^2_{\mathit{err}}$ be the error variance of $\mathit{RLWE}'(\boldsymbol{m})$. Under the constraint $B^d\delta > Q$, the following inequality is satisfied.*

$$\mathit{Var}\Big(\mathit{Err}\big(\boldsymbol{a} \odot \mathit{RLWE}'(\boldsymbol{m})\big)\Big) \leq d \cdot N \cdot B^2/12 \cdot \sigma^2_{\mathit{err}} + \|\boldsymbol{m}\|_2^2 \cdot \delta^2/12.$$

*Proof.* Let $\vec{\boldsymbol{a}}$ be the decomposition of $\boldsymbol{a}$. Then $\boldsymbol{a} \odot \mathsf{RLWE}'(\boldsymbol{m})$ is equal to

$$\begin{aligned}\boldsymbol{a} \odot \mathsf{RLWE}'(\boldsymbol{m}) &= \boldsymbol{a} \odot (\mathsf{RLWE}'(\boldsymbol{0}) + \boldsymbol{m} \cdot \vec{g}_\delta) \\ &= \langle \vec{\boldsymbol{a}}, \mathsf{RLWE}'(\boldsymbol{0})\rangle + \boldsymbol{m} \cdot \langle \vec{\boldsymbol{a}}, \vec{g}_\delta\rangle\end{aligned}$$

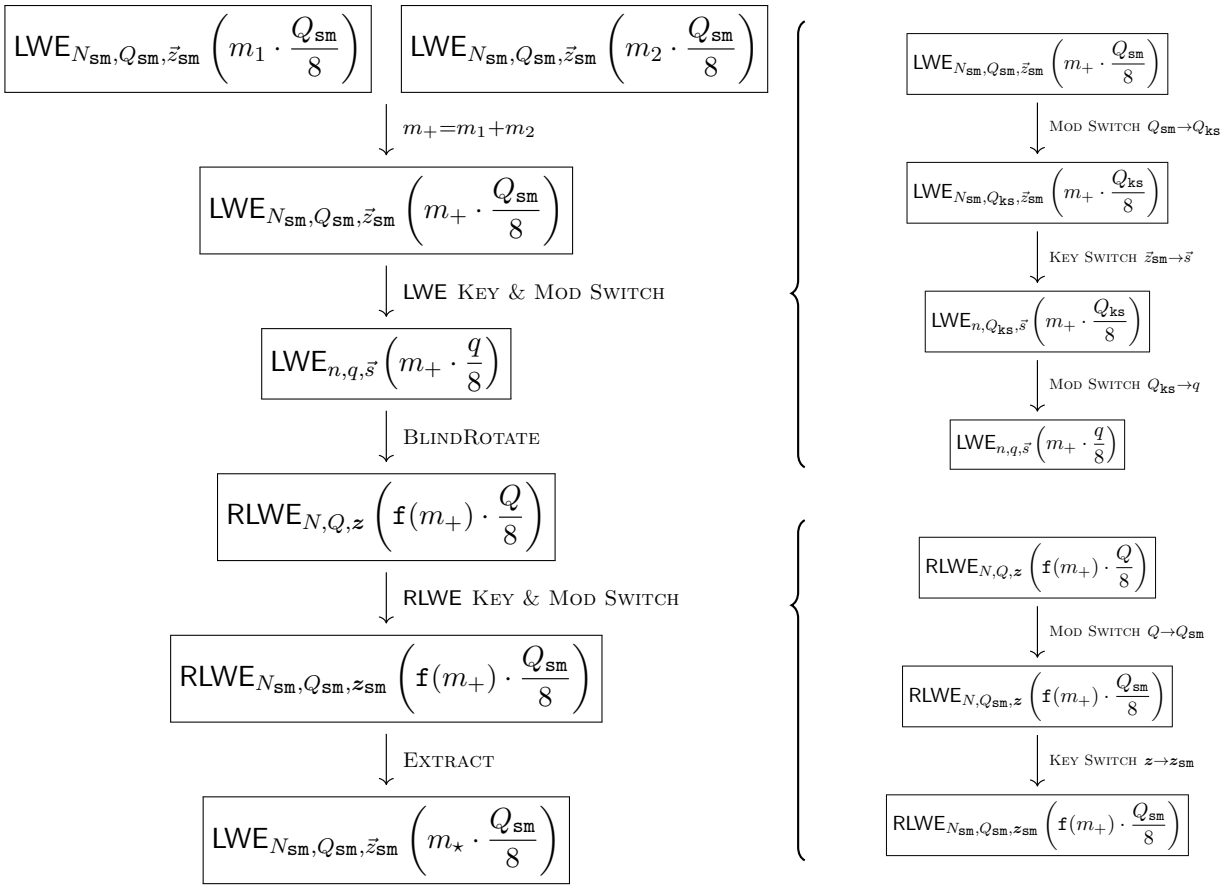


Figure 4: Our variant of bootstrapping for NAND gate

Here, $\langle \vec{a}, \vec{g}_\delta \rangle = \boldsymbol{a} + \boldsymbol{\epsilon}$ where $\|\boldsymbol{\epsilon}\|_\infty = \|\boldsymbol{a} - \langle \vec{a}, \vec{g}_\delta \rangle\|_\infty \leq \delta/2$. Note that $\boldsymbol{a}$, which is part of $\mathsf{RLWE}$ ciphertext, is mostly generated as a uniformly random value, and thus $\boldsymbol{a} - \langle \vec{a}, \vec{g}_\delta \rangle$ introduces an error with variance $\delta^2/12$. Then we have

$$\begin{aligned}\mathsf{Var}\left(\mathsf{Err}\left(\boldsymbol{a} \odot \mathsf{RLWE}'(\boldsymbol{m})\right)\right) &\leq \mathsf{Var}\left(\vec{\boldsymbol{a}} \cdot \mathsf{Err}\left(\mathsf{RLWE}'(\boldsymbol{m})\right)\right) + \mathsf{Var}\left(\boldsymbol{m} \cdot \boldsymbol{\epsilon}\right) \\ &\leq d \cdot N \cdot B^2/12 \cdot \sigma^2_{\mathtt{err}} + \|\boldsymbol{m}\|_2^2 \cdot \delta^2/12.\end{aligned}$$

$\|\boldsymbol{m}\|_2^2$ depends on the message of $\mathsf{RLWE}'$ ciphertext so that we can estimate in advance. In blind rotation, we use $\mathsf{RLWE}'(X^{a_i s_i})$, and $\|X^{a_i s_i}\|_2^2 = 1$. In key switching, $\boldsymbol{m} = \boldsymbol{s}^2$ is a small key for a binary secret key $\boldsymbol{s}$ and thus $\left\|\boldsymbol{s}^2\right\|_2^2 \leq N^2/4$. □

We can also do the similar to $\mathsf{LWE}$ key switching, which has less error and larger key size compared to $\mathsf{RLWE}$ key switching. The $\mathsf{LWE}$ key switching keys contains $dBn$ of $\mathsf{LWE}$ ciphertexts, $\left\{\mathsf{LWE}_{\vec{z}}(kB^i s_j)\right\}_{i\in[0,d-1], j\in[0,n-1], k\in(-B/2,B/2]}$. Proposition 2 is shows the error growth of $\mathsf{LWE}$ key switching with approximate gadget decomposition.

**Proposition 2.** *Let $\sigma^2_{\mathit{err}}$ be the error variance of each $\mathsf{LWE}(kB^i m_j)$. Under the constraint $B^d\delta > Q$, the following inequality is satisfied.*

$$\mathit{Var}\left(\mathit{Err}\left(\sum_{i,j} \mathit{LWE}(h(a_j)_i \cdot B^i m_j)\right)\right) \leq d \cdot N \cdot \sigma^2_{\mathit{err}} + \|\vec{m}\|_2^2 \cdot \delta^2/12,$$

*where $h(a_j)_i$ is the $i$-th element of $h(a_j)$.*

# 4 Analysis & Implementation

## 4.1 Noise Analysis

We follow [CGGI17, MP21, LMK+23] to analyze the total maximum relative variance $\bar{\sigma}_{\mathtt{total}}$. The failure probability can be calculated as $1 - \mathrm{erf}\left(\frac{1}{8\sqrt{2}\bar{\sigma}_{\mathtt{total}}}\right)$.

The parameters $d_{\mathtt{br}}$, $d_{\mathtt{sm}}$, $d_{\mathtt{sk}}$, $d_{\mathtt{ak}}$, $d_{\mathtt{sqk}}$ are corresponting to the number of digits used for blind rotation keys $\mathtt{brk}$, RLWE key switching keys, LWE key switching keys, automorphisms keys $\mathtt{ak}$, and square key $\mathtt{sqk}$ respectively. The parameters $\delta_{\mathtt{br}}$, $\delta_{\mathtt{sm}}$, $\delta_{\mathtt{sk}}$, $\delta_{\mathtt{ak}}$, $\delta_{\mathtt{sqk}}$ are corresponding to the approximate gadget decompositions for the keys. For unrolling techniques we used [BMMP18] and [ZYL+18] methods. The [ZYL+18] method has a larger key size and slightly smaller failure probability compare to [BMMP18] for the same unrolling factor $u$.

The rescaling variances $\sigma^2_{\mathtt{rs},\boldsymbol{x}}$ and $\sigma^2_{\mathtt{rs},\vec{y}}$ depends on the distribution of the underlying secret key $\boldsymbol{x}$ and $\vec{y}$ respectively. For our analysis, we use the following estimates:

- $\sigma^2_{\mathtt{rs},\boldsymbol{x}} = \frac{\|\boldsymbol{x}\|_2^2+1}{12}$ and $\sigma^2_{\mathtt{rs},\vec{y}} = \frac{\|\vec{y}\|_2^2+1}{12}$ - rescaling error for both binary and ternary $\boldsymbol{x}$ and $\vec{y}$

The key is recovered using Algorithm 3. Let $v_\ell$ be the error variance of each ciphertext $\mathtt{res}_i$ in line 4, Algorithm 3. Assuming the error in each coefficient is independent, we have $v_\ell \leq 2 \cdot v_{2\ell} + \sigma^2_{\mathtt{aut}}$ from lines 6 – 9, as the error in some coefficient is canceled out. Solving the recurrence relation, we have $\tilde{\sigma}^2 = v_1 \leq N\sigma^2 + (N-1)\sigma^2_{\mathtt{aut}} \leq N\sigma^2_{\mathtt{aut}}$, where $\sigma^2_{\mathtt{aut}} = d_{\mathtt{aut}} \frac{B^2_{\mathtt{aut}}}{12} N\sigma^2 + \frac{\delta^2_{\mathtt{aut}}}{12} \cdot \|\boldsymbol{z}\|_2^2$ is variance of introduced by automorphism. Then, we find $\mathsf{RLWE}'(s_i \cdot \boldsymbol{z})$ using EvalSquareMult

(Equation (1)) to recover $\mathsf{RGSW}(s_i) = \left\{\mathsf{RLWE}'(s_i), \mathsf{RLWE}'(s_i \cdot \boldsymbol{z})\right\}$. Error variance of $\mathsf{RLWE}'(s_i \cdot \boldsymbol{z})$ is $\hat{\sigma}^2 = \tilde{\sigma}^2 \cdot \|\boldsymbol{z}\|_2^2 + d_{\mathtt{sq}} \frac{B_{\mathtt{sq}}^2}{12}\sigma^2 + \frac{\delta_{\mathtt{sq}}^2}{12} \cdot \left\|\boldsymbol{z}^2\right\|_2^2$.

The error variance for our variant of packing can be estimated as:

- $\tilde{\sigma}^2 \leq N\left(d_{\mathtt{aut}} \frac{B_{\mathtt{aut}}^2}{12} N\sigma^2 + \frac{\delta_{\mathtt{aut}}^2}{12} \cdot \|\boldsymbol{z}\|_2^2\right)$ - for variance of $\mathsf{RLWE}'(s_i)$
- $\hat{\sigma}^2 = \tilde{\sigma}^2 \cdot \|\boldsymbol{z}\|_2^2 + d_{\mathtt{sq}} \frac{B_{\mathtt{sq}}^2}{12}\sigma^2 + \frac{\delta_{\mathtt{sq}}^2}{12} \cdot \left\|\boldsymbol{z}^2\right\|_2^2$ - for variance of $\mathsf{RLWE}'(s_i \cdot \boldsymbol{z})$
- $\sigma_{\mathtt{br}}^2 = \begin{cases} 2n \cdot \left(2d_{\mathtt{br}} \frac{B_{\mathtt{br}}^2}{12} N(\tilde{\sigma}^2 + \hat{\sigma}^2) + \frac{\delta_{\mathtt{br}}^2}{12} \cdot (\|\boldsymbol{z}\|_2^2 + 1)\right) & \text{- for CGGI variant blind rotation} \\ \frac{2n \cdot (2^u - 1)}{u} \cdot \left(2d_{\mathtt{br}} \frac{B_{\mathtt{br}}^2}{12} N(\tilde{\sigma}^2 + \hat{\sigma}^2) + \frac{\delta_{\mathtt{br}}^2}{12} \cdot (\|\boldsymbol{z}\|_2^2 + 1)\right) & \text{- for BMMP variant blind rotation} \\ \frac{n \cdot 2^u}{u} \cdot \left(2d_{\mathtt{br}} \frac{B_{\mathtt{br}}^2}{12} N(\tilde{\sigma}^2 + \hat{\sigma}^2) + \frac{\delta_{\mathtt{br}}^2}{12} \cdot (\|\boldsymbol{z}\|_2^2 + 1)\right) & \text{- for ZYLZD variant blind rotation} \end{cases}$
- $\sigma_{\mathtt{ks}_1}^2 = d_{\mathtt{sm}} \frac{B_{\mathtt{sm}}^2}{12} N\sigma^2 + \frac{\delta_{\mathtt{sm}}^2}{12} \cdot \|\boldsymbol{z}\|_2^2$ - for key switching from $\boldsymbol{z}$ to $\boldsymbol{z}_{\mathtt{sm}}$
- $\sigma_{\mathtt{ks}_2}^2 = d_{\mathtt{ks}} \frac{B_{\mathtt{ks}}^2}{12} N_{\mathtt{sm}}\sigma^2 + \frac{\delta_{\mathtt{ks}}^2}{12} \cdot \|\vec{z}_{\mathtt{sm}}\|_2^2$ - for key switching from $\vec{z}_{\mathtt{sm}}$ to $\vec{s}$.

The total estimate of the relative variance is:

$$\bar{\sigma}_{\mathtt{total}}^2 = \frac{1}{Q^2} 2\sigma_{\mathtt{br}}^2 + \frac{1}{Q_{\mathtt{sm}}^2}(2\sigma_{\mathtt{rs},\boldsymbol{z}}^2 + 2\sigma_{\mathtt{ks}_1}^2) + \frac{1}{Q_{\mathtt{ks}}^2}(\sigma_{\mathtt{rs},\vec{z}_{\mathtt{sm}}}^2 + \sigma_{\mathtt{ks}_2}^2) + \frac{1}{q^2}\sigma_{\mathtt{rs},\vec{s}}^2$$

### 4.2 Parameter Selection & Implementation Results

All the parameter sets for Table 2 were chosen to be at least 128 bits of security and failure probability less than $2^{-32}$. The parameters `128_bCGGI`, `128_tCGGI` and `128_gLMKCDEY` are corresponding to the binrary CGGI, ternary CGGI and gaussian LMKCDEY schemes considered in [LMK+23] as state of the art schemes. For our paramter sets `128_bCGGI_Our`, `128_bBMMP_u2_Our`, `128_bZYLZD_u2_Our` we used CGGI, BMMP and ZYLZD schemes with binary secrets respectively.

In order to provide a fair comparison we have implemented all algorithms using the OpenFHE library (v.1.1.0). The evaluation environment is Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz, running Ubuntu 20.04.3 LTS. We compiled with clang 10 and the following CMake flags: `WITH_OPENMP=OFF`, `WITH_NATIVEOPT=ON`, `-DNATIVE_SIZE=64`. We also provide implementation results, compiled with `-DNATIVE_SIZE=32` instead of `-DNATIVE_SIZE=64` as was proposed in [MP21, LMK+23] for efficiency.

**Remark 4.** *We did not consider Torus variants for comparison in order to keep the same environment. For the same reasons we have not considered NTRU variants, but it is of great interest to adapt our techniques to NTRU and Torus variants of bootstrapping, find suitable parameters, and do full a comparison.*

Both the original and our variant utilize a common reference string model (CRS) to generate the $\boldsymbol{a}$ and $\vec{a}$ components of each transferred RLWE and LWE ciphertext. This enables us to transmit only the $\boldsymbol{b}$ and $\vec{b}$ components to the server. Upon receiving the $\boldsymbol{b}$ and $\vec{b}$ components, the server

| Params | $\log Q$ | $\log Q_{\mathtt{sm}}$ | $\log Q_{\mathtt{ks}}$ | $\log q$ | $N$ | $N_{\mathtt{sm}}$ | $n$ | $d_{\mathtt{br}}$ | $d_{\mathtt{sm}}$ | $d_{\mathtt{ks}}$ | $d_{\mathtt{ak}}$ | $d_{\mathtt{sqk}}$ | $u$ | Security |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128_tGINX[LMK+23] | 26 | - | 14 | 11 | $2^{10}$ | - | 531 | 4 | - | 2 | - | - | - | $2^{-128.5}$ |
| 128_bGINX[LMK+23] | 25 | - | 14 | 11 | $2^{10}$ | - | 571 | 4 | - | 2 | - | - | - | $2^{-128.1}$ |
| STD128_gLMKCDEY [LMK+23] | 28 | - | 14 | 11 | $2^{10}$ | - | 458 | 3 | - | 2 | - | - | - | $2^{-128.2}$ |
| 128_bCGGI_Our | 54 | 27 | 14 | 11 | $2^{11}$ | $2^{10}$ | 571 | 3 | 2 | 3 | 5 | 2 | - | $2^{-128.1}$ |
| 128_bBMMP_u2_Our | 54 | 27 | 14 | 11 | $2^{11}$ | $2^{10}$ | 571 | 3 | 2 | 3 | 5 | 2 | 2 | $2^{-128.1}$ |
| 128_bZYLZD_u2_Our | 54 | 27 | 14 | 11 | $2^{11}$ | $2^{10}$ | 571 | 3 | 2 | 3 | 5 | 2 | 2 | $2^{-128.1}$ |

Table 2: Parameter sets used in the implementation.

| Parameters set | Key Generation time, ms | Transfer key size | Reconstruct time, ms | Bootstrapping key size | Bootstrapping time, ms | Failure probability |
|---|---|---|---|---|---|---|
| 128_tCGGI [LMK+23] | 6087 (5423) | 28.83 Mb | - | 233 Mb | 123 (79) | $2^{-35.79}$ |
| 128_bCGGI [LMK+23] | 5598 (5932) | 16.48 Mb | - | 250 Mb | 108 (74) | $2^{-37.26}$ |
| STD128_gLMKCDEY [LMK+23] | 4435 (4314) | 9.01 Mb | - | 218 Mb | 99 (70) | $2^{-43.51}$ |
| 128_bCGGI_Our | 873 | 881 Kb | 815 | 175 Mb | 202 | $2^{-44.88}$ |
| 128_bBMMP_u2_Our | 875 | 894 Kb | 1689 | 220 Mb | 147 | $2^{-35.71}$ |
| 128_bZYLZD_u2_Our | 877 | 894 Kb | 1790 | 265 Mb | 169 | $2^{-44.88}$ |

Table 3: Implementation results. Values in brackets correspond to the `-DNATIVE_SIZE=32` compilation setting

reconstructs the $\boldsymbol{a}$ and $\vec{a}$ components for each ciphertext using the same CRS. We also do not take into account the server time to reconstruct the CRS parts.

We successfully achieved a significant reduction in the size of transferred keys, reducing it to less than a Megabyte. While our implementation results show that the bootstrapping time has increased by approximately $\times 2$ compared to the original method, it is important to note that this increase is primarily attributed to the use of the `-DNATIVE_SIZE=32` settings in the original method.

In addition, despite the higher parameters used in our method, the total time required for key generation and reconstruction in our variant is occasionally smaller compared to the original "non-packed" method. We explain this phenomenon by the fact that our approach uses a smaller number of digits for $\mathsf{RLWE}'$ keys and uses fewer number Number Theoretic Transforms (NTTs) during the process of generating and reconstructing keys in our approach, compared to the original method.

# 5 Conclusion

The size of bootstrapping keys can pose a significant challenge for real-world problems, and it is crucial not to ignore this issue. Despite the overhead costs in bootstrapping time associated with increased parameters in our method, our approach of packing keys demonstrates a substantial reduction in key size for CGGI, BMMP, and ZYLZD schemes. It would be valuable to conduct further research on the application and comparison of our key packing technique to other FHEW-like schemes, such as the Torus variant TFHE [CGGI20], NTRU variant FINAL [BIP+22], and automorphism variant LMKCDEY [LMK+23]. Our future plans also involve exploring the potential of applying our key packing technique to BGV/BFV/CKKS schemes.

# References

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *CRYPTO 2014*, pages 297–314. Springer, 2014.

[BCC+22] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. Meta-bts: Bootstrapping precision beyond the limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 223–234, 2022.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BIP+22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE instantiated with NTRU and LWE. *Cryptol. ePrint Arch.*, 2022/074, 2022.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology – CRYPTO 2018*, pages 483–512. Springer, 2018.

[BMTH21] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Advances in Cryptology – EUROCRYPT 2021*. Springer, 2021.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In *Advances in Cryptology – CRYPTO 2011*, pages 505–524. Springer, 2011.

[CDKS21] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (Ring) LWE ciphertexts. In *Applied Cryptography and Network Security*. Springer, 2021.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *Advances in Cryptology – ASIACRYPT 2017*, pages 377–408. Springer, 2017.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[CHK+21] Jihoon Cho, Jincheol Ha, Seongkwang Kim, Byeonghak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. Transciphering framework for approximate homomorphic encryption. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III*, pages 640–669. Springer, 2021.

[CIL21] Hao Chen, Ilia Iliashenko, and Kim Laine. When HEAAN meets FV: a new somewhat homomorphic encryption with reduced memory overhead. In *Cryptography and Coding: 18th IMA International Conference, IMACC 2021*, pages 265–285. Springer, 2021.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437. Springer, 2017.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT 2015*, pages 617–640. Springer, 2015.

[GdMRT23] Francisco AA Gomes, Filipe de Matos, Paulo Rego, and Fernando Trinta. Analysis of the impact of homomorphic algorithm on offloading of mobile application tasks. In *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, pages 961–962. IEEE, 2023.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM Symposium on Theory of Computing*, pages 169–178. ACM, 2009.

[GINX16] Nicolas Gama, Malika Izabachene, Phong Q Nguyen, and Xiang Xie. Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. In *EUROCRYPT 2016*, pages 528–558. Springer, 2016.

[KDE+21] Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for RLWE-based homomorphic encryption. *Cryptol. ePrint Arch.*, 2021/691, 2021.

[KPZ21] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting homomorphic encryption schemes for finite fields. In *Advances in Cryptology – ASIACRYPT 2021*, pages 608–639. Springer, 2021.

[LHH+21] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE symposium on Security and Privacy (S&P)*, pages 1057–1073. IEEE, 2021.

[LLKN22] Joon-Woo Lee, Eunsang Lee, Young-Sik Kim, and Jong-Seon No. Hierarchical galois key management systems for privacy preserving aiaas with homomorphic encryption. *Cryptology ePrint Archive*, 2022.

[LMK+23] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In *Advances in Cryptology – EUROCRYPT 2023*, pages 227–256. Springer, 2023.

[MP21] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like cryptosystems. In *WAHC'21*, pages 17–28. ACM, 2021.

[NLV11] 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.

[Ope22] OpenFHE. Open-Source Fully Homomorphic Encryption Library. `https://github.com/openfheorg/openfhe-development`, 2022.

[PCFBC19] Goiuri Peralta, Raul G Cid-Fuentes, Josu Bilbao, and Pedro M Crespo. Homomorphic encryption and network coding in iot architectures: Advantages and future challenges. *Electronics*, 8(8):827, 2019.

[RTD+21] Wang Ren, Xin Tong, Jing Du, Na Wang, Shan Cang Li, Geyong Min, Zhiwei Zhao, and Ali Kashif Bashir. Privacy-preserving using homomorphic encryption in mobile iot systems. *Computer Communications*, 165:105–111, 2021.

[SK19] Rakesh Shrestha and Shiho Kim. Integration of iot with blockchain and homomorphic encryption: Challenging issues and opportunities. In *Advances in computers*, volume 115, pages 293–331. Elsevier, 2019.

[ZYL+18] Tanping Zhou, Xiaoyuan Yang, Longfei Liu, Wei Zhang, and Ningbo Li. Faster bootstrapping with multiple addends. *IEEE Access*, 6:49868–49876, 2018.