Practical Randomized Lattice Gadget Decomposition With Application to FHE

Sohyun Jeon[1], Hyang-Sook Lee[1], and Jeongeun Park[2]

[1] Department of Mathematics, Ewha Womans University, Seoul, Republic of Korea
jeonsh099@ewhain.net, hsl@ewha.ac.kr
[2] imec-COSIC, KU Leuven, Leuven, Belgium
jeongeun.park@esat.kuleuven.be

Abstract. Gadget decomposition is widely used in lattice based cryptography, especially homomorphic encryption (HE) to keep the noise growth slow. If it is randomized following a subgaussian distribution, it is called subgaussian (gadget) decomposition which guarantees that we can bound the noise contained in ciphertexts by its variance. This gives tighter and cleaner noise bound in average case, instead of the use of its norm. Even though there are few attempts to build efficient such algorithms, most of them are still not practical enough to be applied to homomorphic encryption schemes due to somewhat high overhead compared to the deterministic decomposition. Furthermore, there has been no detailed analysis of existing works. Therefore, HE schemes use the deterministic decomposition algorithm and rely on a Heuristic assumption that every output element follows a subgaussian distribution independently.
In this work, we introduce a new practical subgaussian gadget decomposition algorithm which has the least overhead (less than 14%) among existing works for certain parameter sets, by combining two previous works. In other words, we bring an existing technique based on an uniform distribution to a simpler and faster design (PKC' 22) to exploit parallel computation, which allows to skip expensive parts due to pre-computation, resulting in even simpler and faster algorithm. When the modulus is large (over 100-bit), our algorithm is not always faster than the other similar work. Therefore, we give a detailed comparison, even for large modulus, with all the competitive algorithms for applications to choose the best algorithm for their choice of parameters.

Keywords: Subgaussian Decomposition, Randomized Gadget Decomposition, Homomorphic Encryption

1 Introduction

Gadget decomposition algorithm is an essential building block for lattice based cryptography which leads to various applications such as identity based encryption (IBE) [GPV08, CM15], attributed based encryption (ABE) [DDP$^+$18, BGG$^+$14], homomorphic encryption (HE) [GSW13] and more. A Gadget matrix is defined as $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}$, where $\mathbf{g} := (1, b, b^2, \ldots, b^{k-1})$ is called a gadget vector, and $\mathbf{I}_n$ is a n-by-n identity matrix for some positive integer n. A gadget decomposition algorithm was firstly introduced by Micciancio et al. [MP12] as preimage sampling for $f_{\mathbf{G}}(\mathbf{x}) = \mathbf{G}\mathbf{x} \mod q$. For an input $\mathbf{u}$, the algorithm samples a point $\mathbf{x}$ in $\Lambda_{\mathbf{u}}^{\perp}(\mathbf{G})$ which is a coset of $\Lambda_q^{\perp}(\mathbf{G})$. We consider the case $n = 1$ so that the algorithm samples a point in $\Lambda_u^{\perp}(\mathbf{g}^t)$ for an input $u \in \mathbb{Z}_q$. Depending on a distribution which the output follows, applications may differ. Specifically, if the output $\mathbf{x}$ is a subguassian random variable, we call it a subgaussian gadget decomposition (subgaussian sampling for short, throughout this paper).

The subgaussian distribution has an important role in lattice based cryptosystems due to its Pythagorean additivity. Informally, any distribution of which tails are bounded by tails of a Gaussian distribution is a subgaussian distribution. Therefore, a discrete Gaussian distribution also belongs to a subgaussian distribution. In particular, the property, Pythagorean additivity, enables to tightly analyze the noise growth of average case in many lattice based homomorphic encryption (HE) schemes like [BGV12, FV12, GSW13, DM15, CGGI20, BIP$^+$22]. A ciphertext of HE schemes has the noise term which becomes larger whenever homomorphic evaluation is performed, leading to decryption failure if it is not refreshed at some point. Therefore, Gentry et.al. [GSW13] firstly introduce the use of the gadget decomposition to keep the noise growth small, hence, their scheme allows more operations before decryption failure occurs. Towards more practical use, there have been many HE schemes [DM15, CGGI20, BIP$^+$22] which basically were built on top of this strategy, and they have been called GSW-like schemes in the related literatures. If such schemes use a *randomized* gadget decomposition, there are more advantages for them: 1) one can analyze the noise contained in ciphertexts with cleaner and tighter bound than the use of other measures such as Euclidean/infinite norm [AP14], and 2) circuit privacy can be achieved almost for free [BdMW16]. That is why we need practical randomized gadget decomposition since FHE schemes and their applications are becoming more practical.

Analyzing the noise growth precisely as much as possible in homomorphic encryption is highly important since the noise growth is closely related to choosing the right parameters of applications based on HE schemes to achieve the best performance. Moreover, one can estimate how many homomorphic operations are possible before decryption fails based on the analysis. More importantly, the parameters of HE schemes determine the bit security of the schemes based on well known attacks. Therefore, the noise analysis can be a tool to justify their choice of parameters which makes the schemes safe from the attacks as in [BIP$^+$22].

1.1 Subgaussian sampling in homomorphic encryption

GSW-like schemes [GSW13, DM15, CGGI16, BIP$^+$22] which implement gate operations consisting of linear operation over ciphertexts can model their noise coefficient as subgaussian random variable due to linearity. Hence, Ducas and Micciancio [DM15] started to use subgaussian analysis to estimate how much the noise grows after evaluating a complicated circuit on average. Nevertheless, the follow-up schemes heuristically assume that their final noise elements independently follow a subgaussian distribution (called independence Heuristic in [CGGI16]), then use a deterministic gadget decomposition in their implementation for the sake of practical performance[3]. The main reason is that 1) subgaussian sampling was studied only in a theoretical way previously so that such work did not receive much attention in practical fields and 2) the only existing algorithm they could employ for their randomized decomposition algorithm was discrete Gaussian sampling [GM18, MP12] which might cause huge computational overhead in implementations. Afterwards, all the follow-up works and applications based on HE keep relying on the Heuristic assumption and using the deterministic algorithm in their implementations. Therefore, it had seemed that this was the only solution to achieve both practicality and such tight noise analysis until Genise et al. [GMP19] presented the first efficient randomized digit decomposition, recently. We denote their algorithm by `GMP19` in this paper.

They focus on subgaussian sampling itself which can be implemented more efficiently than the discrete Gaussian sampling in practice mainly due to its relaxed probability condition. As a result, they presented the first subgaussian sampling which outperforms existing discrete Gaussian sampling, so that it became closer to practical algorithm for GSW-like schemes [GSW13, DM15, CGGI16, BIP$^+$22]. Despite of their efforts, the computational overhead, which is the extra running time after running deterministic decomposition algorithm, is not negligible. Later, Jeon, Lee, and Park [JLP21] observed that the main two subalgorithms of `GMP19` were sequential so that they parallelized the two with a uniform distribution and showed that their uniform distribution is subgaussian. With this approach, one algorithm can be considered as a pre-computation, hence, their solution performs over 50% better than `GMP19`.

Moreover, Zhang and Yu [ZY22] also improved `GMP19` when q is not a power of b, introducing a plausible idea by calling the simpler algorithm of [GMP19] for $q = b^k$ for some positive integer k as a subalgorithm. In more detail, Genise et al. presented two different subgaussian samplings depending on the relation between q and b due to different basis structures of the lattice. The algorithm when $q \neq b^k$ has more complicated steps than the other one for $q = b^k$, hence it takes more time than the other. Zhang and Yu focused on the similarity of the two bases. In other words, the two bases look exactly same up to the $(k-1)$-th column, and the last column of them only differs. Therefore, they run the faster algorithm to obtain the result up to the $(k-1)$-th digit of the final result by reducing the modulus q such that $b^{k-1} < q < b^k$ to $q' = b^{k-1}$, then determine the last digit by checking all the previous outcome. Due to the use of simpler and faster

[3] Note that the deterministic gadget decomposition takes a uniform random ciphertext, hence its output follows a uniform random distribution.

algorithm, they could have better computation time than GMP19. However, the algorithm is still not practical enough in terms of the actual computation time. In reality, the main computation overhead of evaluating a homomorphic circuit would be caused by this randomized gadget decomposition algorithm. Furthermore, no detailed comparison between the two different techniques [ZY22, JLP21], both of which outperform [GMP19], has been addressed in any literature. In fact, it is important to compare these existing algorithms to see the trade-offs in different parameter settings for those who is interested in HE and its applications. It is because the performance of the gadget decomposition and the noise growth of the output highly depend on the choice of b and k for a fixed q. In detail, the larger k, the lower noise is added to the output ciphertext, but also the slower computation time the algorithm has.

1.2 Our Contribution

In this work, we present a faster randomized gadget decomposition (subgaussian sampling) with the least computation overhead compared to the deterministic decomposition among the existing works. We bring the technique of Zhang-Yu [ZY22](denoted by ZY22 for short) to the subgaussian sampling of Jeon-Lee-Park [JLP21] (denoted by JLP21) which uses a uniform distribution, so that we can fully exploit the advantage of precomputation of JLP21 in the structure of ZY22. In other words, we replace the call of GMP19 by the call of JLP21 in Zhamg-Yu's structure, so we could already gain a little improvement because JLP21 is faster than GMP19. And the last step of this algorithm, which checks all the previous outcome to determine the last digit, becomes simpler than ZY22 since we can skip this step by checking the only one precomputed value. Consequently, our results range between 5x to 14x faster compared to Zhang-Yu's.

In addition, we give a detailed analysis and comparison among the existing such algorithms [GMP19, ZY22, JLP21], which has not been covered in the previous literature. In more detail, our experimental result shows that our algorithm outperforms ZY22 and slightly faster than JLP21 for small q ($\approx 2^{60}$). We note that the value k increases as b decreases for a fixed q. Our algorithm is 82% faster than ZY22 at most and 35% faster than JLP21 with the large k. JLP21 becomes similar to ours as k gets smaller since one of its subalgorithms which depends on k becomes faster than one of ours which takes constant time. Also, we have the least computational overhead (from 2% to 14% depending on k) among existing works, which means that it takes only a little bit longer time than deterministic algorithm.

For larger q such as $q \approx 2^{102}$, the computation cost of four different algorithms are almost same due to the use of BigInteger type which represents a number over 64-bit in implementation. However, both JLP21 and our algorithm are slightly faster thanks to the uniform distribution. When q is large and k is small, JLP21 outperforms our algorithm due to the same reason as q is small. Hence, it is suitable for applications which require low multiplication depth when the modulus q is large. We note that most of applications of HE which require large q such as [BGV12, CKKS17] use CRT/RNS technique to avoid multi-precision as discussed in [GMP19], so that the result with smaller q would be more helpful for such applications.

We also note that the tighter bound of the noise is still preserved when non-centered distribution is used in HE since the extra term is much smaller than the dominant term in the variance (discussed in Section 4.2). Therefore, the non-centered case has slightly larger size of output, so does the noise in ciphertexts of HE, but it does not directly influence on the most significant bit of the noise.

1.3 Technical Overview

Let's say that we want to obtain a decomposition of $u \in \mathbb{Z}_q$ where $q \neq b^k$, given a gadget vector $\mathbf{g} = (1, b, b^2, \ldots, b^{k-1})$. JLP21 works as follows: 1) sample a vector $\mathbf{y}$ uniformly at random and 2) compute $(x_0, \ldots, x_{k-1}) =: \mathbf{x} = \mathbf{S}_q\mathbf{y} + \mathbf{u}$, where $\mathbf{u}$ is a deterministic digit decomposition of u. Since sampling $\mathbf{y}$ is totally independent of the input u in their algorithm, this step can be computed previously as a preprocessing. We observed that the last component of $\mathbf{y}$, say y_{k-1}, determines if $\mathbf{x}$ is a decomposition of u or $u - q$ in JLP21 since y_{k-1} is a coefficient of the last column of the basis. Therefore, the algorithm already knows that whether the composed value will be u or $u - q$ by checking the precomputed value y_{k-1}.

Then, we let the algorithm fix a value denoted by u' for depending on y_{k-1}, that is, $u' = u \mod b^{k-1}$ if $y_{k-1} = 0$ and $u' = u - q \mod b^{k-1}$ if $y_{k-1} = -1$. Next, we use the trick of ZY22 to compute from x_0 to x_{k-2}

of $\mathbf{x}$ by running the subgaussian sampling of `JLP21` for power of b case taking u' and $q' = b^{k-1}$ on input. Now, it is time to decide the last component of $\mathbf{x}$, x_{k-1}. As [ZY22] observed already, the last component x_{k-1} is determined by the value of $\langle \mathbf{x}', \mathbf{g}' \rangle$, where $\mathbf{x} = (\mathbf{x}', x_{k-1})$ and $\mathbf{g} = (\mathbf{g}', b^{k-1})$.

Due to sequential process of `ZY22`, it is necessary to compute the dot product, however, we can already check the value by checking the second last component of the precomputed vector $\mathbf{y}$, y_{k-2}, by our observation of `JLP21` structure. Consequently, we can more quickly determine the last component of $\mathbf{x}$ than the previous work.

2 Preliminaries

Notation: Numbers are denoted as small letters, such as $a \in \mathbb{Z}$, vectors as bold small letters, $\mathbf{a} \in \mathbb{Z}^n$, and matrices as capital bold letters, $\mathbf{A} \in \mathbb{R}^{n \times n}$. We denote the inner product of two vectors $\mathbf{v}, \mathbf{w}$ by $\langle \mathbf{v}, \mathbf{w} \rangle$. We use the ℓ_2 norm as a default norm for a vector $\mathbf{x}$. $[u]_b^k = (u_0, \ldots, u_{k-1})$ denotes a vector, where $u_i \in \{0, \ldots, b-1\}$, which is b-ary decomposition of u such that $\sum_i b^i u_i = u$ for an integer base $b > 0$. A notation $a \xleftarrow{\$} S$ means that a is chosen uniformly from a set S.

2.1 Subgaussian Random Variables

We explain subgaussian random variables and their significant properties in this section. We describe the general definition for a univariate δ-subgaussian random variable for some $\delta \geq 0$ as in [MP12].

Definition 1. *A random variable V over $\mathbb{R}$ is δ-subgaussian ($\delta \geq 0$) with parameter $s > 0$ if its moment generating function satisfies*

$$\mathbb{E}[\exp(2\pi t V)] \leq \exp(\delta) \exp(\pi s^2 t^2)$$

for all $t \in \mathbb{R}$.

We call the parameter s the standard parameter. It is easy to see that if X is δ-subgaussian with parameter s, then cX is also δ-subgaussian with parameter $|c|s$ for any $c \in \mathbb{R}$. In addition, if a random variable V is centered at 0 and bounded with B, then V is 0-subgaussian with parameter $B\sqrt{2\pi}$ [Str94]. If there exists a 0-subgaussian random variable, we can make it into a δ-subgaussian random variable for nonzero δ by shifting the variable with some real number as stated in Lemma 1. Therefore, we can deal with δ-subgaussian distribution by considering a shifted centered subgaussian distribution.

Lemma 1. (Lemma 7 in [MP19]) *If $\overline{V}$ is a 0-subgaussian with parameter $\overline{s}$, then the real-valued shifted random variable $V = \overline{V} + \alpha$ for $\alpha \in \mathbb{R}$ is a δ-subgaussian with parameter s such that $s > \overline{s}$ for some non-negative real-valued δ such that $\delta \geq \alpha^2\pi/(s^2 - \overline{s}^2)$.*

In other words, a subgaussian distribution centered at nonzero is δ-subgaussian where $\delta > 0$. Informally speaking, a distribution is more centered at 0 if δ is closer to 0.

Lemma 2 says that $\delta(> 0)$-subgaussian random variable with parameter s has variance bounded by s^2. Informally, the tails of V are dominated by a Gaussian function with standard deviation s.

Lemma 2. (Lemma 8 in [MP19]) *If V is a univariate real-valued δ-subgaussian with parameter $s \geq 0$, then $\mathsf{Var}(V) \leq s^2$, where $\mathsf{Var}(V)$ is the variance of V.*

The sum of independent subgaussian variables is easily seen to be subgaussian. It is also proved that the sum of subgaussian variables is also subgaussian even when random variables are conditioned on the other random variables. And we use this property to prove that our algorithm follows a subgaussian distribution.

Lemma 3. (Claim 2.1 in [LPR13]) *Let $\delta_i, s_i \geq 0$ and X_i be random variables for $i = 1, \ldots, k$. Suppose that for every i, when conditioning on any values of $X_1, \ldots, X_{i-1}$, the random variable X_i is δ_i-subgaussian with parameter s_i. Then $\sum X_i$ is $\sum \delta_i$-subgaussian with parameter $\sqrt{\sum s_i^2}$.*

Using Lemma 3, we can prove that a vector with subgaussian coordinates is also subgaussian, which is a general extension from Lemma 2.2 in [GMP19]. To do this, we use the fact that a random vector $\mathbf{x} \in \mathbb{R}^n$ is δ-subgaussian with parameter $s > 0$ if $\langle \mathbf{x}, \mathbf{u} \rangle$ is δ-subgaussian with parameter s for all unit vectors $\mathbf{u}$, given in [MP12].

Lemma 4. (The general version of Lemma 2.2 in [GMP19]) *Let $\mathbf{x}$ be a discrete random vector over $\mathbb{R}^n$ such that each coordinate x_i is δ_i-subgaussian with parameter s_i given the previous coordinates take any values. Then $\mathbf{x}$ is a $\sum \delta_i$-subgaussian vector with parameter $\max_i\{s_i\}$.*

Proof. The moment generating function of $\langle \mathbf{x}, \mathbf{u} \rangle$ is

$$\begin{aligned}
\mathbb{E}[\exp(2\pi t \langle \mathbf{x}, \mathbf{u} \rangle)] &= \mathbb{E}[\exp(2\pi t \sum x_i u_i)] \\
&\le \exp(\sum \delta_i) \exp(\pi t^2 \sum s_i^2 u_i^2) (\because \text{Lemma } 3) \\
&\le \exp(\sum \delta_i) \exp(\pi t^2 (\max s_i)^2 \sum u_i^2) \\
&= \exp(\sum \delta_i) \exp(\pi t^2 (\max s_i)^2 ||\mathbf{u}||^2) \\
&= \exp(\sum \delta_i) \exp(\pi t^2 (\max s_i)^2 (\because \text{unit vector } \mathbf{u}).
\end{aligned}$$

□

2.2 Gadget and Lattices

We use the same gadget $\mathbf{g} = (1, b, \ldots, b^{k-1})$ defined in [MP12] for a positive integer b. A lattice Λ with the rank k and basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ is a set of all linear combinations of the basis vectors with coefficients in $\mathbb{Z}$. A coset of a lattice Λ is a set $\mathbf{c} + \Lambda = \{\mathbf{c} + \mathbf{z} : \mathbf{z} \in \Lambda\}$. In this work, we focus on the gadget lattice $\Lambda_q^\perp(\mathbf{g}^t) = \{\mathbf{z} \in \mathbb{Z}_q^k : \langle \mathbf{g}, \mathbf{z} \rangle = 0 \mod q\}$ for $q \le b^k$. For any $u \in \mathbb{Z}_q$, $\Lambda_u^\perp(\mathbf{g}^t) = \{\mathbf{z} \in \mathbb{Z}^k : \langle \mathbf{g}, \mathbf{z} \rangle = u \mod q\}$ is a coset of $\Lambda_q^\perp(\mathbf{g}^t)$ since $\Lambda_u^\perp(\mathbf{g}^t) = \mathbf{u} + \Lambda_q^\perp(\mathbf{g}^t)$ where $\mathbf{u}$ is a vector such that $\langle \mathbf{g}, \mathbf{u} \rangle = u \mod q$. A basis of the gadget lattice $\Lambda_q^\perp(\mathbf{g}^t)$ is like the following:

$$\begin{aligned}
\mathbf{S}_q &= \begin{bmatrix} b & & & & q_0 \\ -1 & b & & & q_1 \\ & & \ddots & & \vdots \\ & & & b & q_{k-2} \\ & & & -1 & q_{k-1} \end{bmatrix} = \left[\begin{array}{c|c} \mathbf{S}' & \\ \hline \quad -1 & \mathbf{q} \end{array}\right] \\
&= \begin{bmatrix} b & & & \\ -1 & b & & \\ & \ddots & & \\ & & b & \\ & & -1 & b \end{bmatrix} \begin{bmatrix} 1 & & & & d_0 \\ & 1 & & & d_1 \\ & & \ddots & & \vdots \\ & & & 1 & d_{k-2} \\ & & & & d_{k-1} \end{bmatrix} = \mathbf{SD} \in \mathbb{Z}^{k \times k},
\end{aligned}$$

where $\mathbf{q}$ is a b-decomposition of q, and $\mathbf{S}' \in \mathbb{Z}^{(k-1)\times(k-1)}$ and $\mathbf{S} \in \mathbb{Z}^{k \times k}$ is a basis of the gadget lattice for $q = b^{k-1}$ and $q = b^k$ respectively. The efficiency of the algorithm highly depends on the structure of the basis of the gadget lattice. When $q < b^k$, which is the general case, the basis $\mathbf{S}_q$ looks similar to $\mathbf{S}$, but has additional elements on its last column, hence the sampling algorithm is more complicated than the special case when q is a power of b. Therefore, [GMP19] uses the factorization $\mathbf{S}_q = \mathbf{SD}$ where $\mathbf{S}$ and $\mathbf{D}$ are sparse and triangular matrix. And then the algorithm requires the linear transformation.

3 New Practical Subgaussian Sampling

In this section, we present our new gadget subgaussian decomposition algorithm for the general case when $q < b^k$. We substitute the call of GMP19 in Zhang- Yu's structure by the call of JLP21 when the modulus q is reduced to $q' = b^{k-1}$.

$$\begin{aligned}
\mathbf{x} &= \mathbf{u} + \mathbf{S}_q\mathbf{y}\\
&= \begin{bmatrix} u_0 \\ \vdots \\ u_{k-2} \\ u_{k-1}\end{bmatrix} + \begin{bmatrix} b & & & q_0 \\ -1 & \ddots & & \vdots \\ & \ddots & b & q_{k-2} \\ & & -1 & q_{k-1}\end{bmatrix}\begin{bmatrix} y_0\\ \vdots \\ y_{k-2}\\ y_{k-1}\end{bmatrix}\\
&= \begin{bmatrix} u_0 \\ \vdots \\ u_{k-2} \\ u_{k-1}\end{bmatrix} + y_{k-1}\begin{bmatrix}q_0\\ \vdots\\ q_{k-2}\\ q_{k-1}\end{bmatrix} + \begin{bmatrix} b & & \\ -1 & \ddots & \\ & \ddots & b \\ & & -1\end{bmatrix}\begin{bmatrix}y_0\\ \vdots\\ y_{k-2}\end{bmatrix}\\
&= \mathbf{u} + y_{k-1}\mathbf{q} + [\mathbf{S}'\mathbf{y}' | -y_{k-2}]
\end{aligned} \tag{1}$$

We observed that the last component of $\mathbf{y}$, denoted by y_{k-1} actually determines whether the output is going to be a decomposition of u or $u - q$ in JLP21. Here each component of $\mathbf{y}$ is chosen as either -1 or 0 at uniformly random. If $q = b^k$ and y_{k-1} is -1, the last component of $\mathbf{S}_q\mathbf{y}(= \mathbf{S}\mathbf{y})$ has $-b$ term as seen in equation (1) (especially red part of $\mathbf{S}_q$). So when it is composed to an integer in $\mathbb{Z}_q$ with the gadget vector $\mathbf{g}$, it contains $-b^k$ which is $-q$. Moreover, when $q < b^k$ and $y_{k-1} = -1$, it influences on every component of $\mathbf{S}_q\mathbf{y}$ due to the structure of $\mathbf{S}_q$ (see the purple part of the equation (1)). In other words, each component of $\mathbf{S}_q\mathbf{y}$ has a decomposition element q, and then it outputs $-q$ after the inner product with $\mathbf{g}$. Therefore, the last column means that a composition value becomes $u - q$.

Now, we use the above observation for our algorithm to improve the efficiency. Our algorithm firstly samples $\mathbf{y}$ by running Algorithm 1 in advance and employ it for online phase as JLP21 does. Like ZY22, we also sample $\mathbf{x}'$ where $\mathbf{x} := (\mathbf{x}', x_{k-1})$ for the reduced modulus $q' = b^{k-1}$. To do this, we first need to fix u' and a, which are an input and the candidate of x_{k-1}, by checking y_{k-1}. In other words, $u' = u \mod b^{k-1}$ and $a = \lfloor \frac{u}{b^{k-1}} \rfloor$ if $y_{k-1} = 0$, $u' = u - q \mod b^{k-1}$ and $a = \lfloor \frac{u-q}{b^{k-1}} \rfloor$ otherwise. It is because the last component of $\mathbf{y}$ determines if the final output $\mathbf{x}$ is going to be a decomposition of u or $u - q$ due to the structure of the base of $\Lambda_q^{\perp}(\mathbf{g}^t)$ as we observed above.

Next it runs Subgaussian′(Algorithm 2), which is JLP21 for a power-of-base modulus, on input (b, q', u') and the first $k - 1$ components of $\mathbf{y}$. Then we obtain $\mathbf{x}'$ from Subgaussian′. Since the subgaussian sampling is a randomized decomposition, $\mathbf{x}'$ can be such that $\langle \mathbf{g}', \mathbf{x}' \rangle = u'$ or $u' - b^{k-1}$ where $\mathbf{g}' = (1, b, \ldots, b^{k-2})$. It is correct for the reduced modulus b^{k-1} since $u' - b^{k-1} \equiv u' \mod b^{k-1}$. However, what we want to obtain is the vector for the modulus q which is not a power of b. Hence, we should compute the last component with a. To determine the last component of output $\mathbf{x}$, ZY22 verifies whether $\langle \mathbf{g}', \mathbf{x}' \rangle = u'$ or $u' - b^{k-1}$ by computing the dot product. Unlike their approach , we can verify this by only checking y_{k-2} based on the same observation. As a result, we can skip the last step of ZY22 which computes $\langle \mathbf{g}', \mathbf{x}' \rangle$, thus we can quickly determine x_{k-1} based on y_{k-2}.

We show that our algorithm outputs a δ-subgaussian vector with a standard parameter which is slightly larger compared to the previous sequential algorithms in Theorem 1.

Theorem 1. *Let $b, q \in \mathbb{N}$, $k = \lceil \log_b q \rceil$, and $u \in \mathbb{Z}_q$. Then the output vector $\mathbf{x}$ of Algorithm 3 is $\frac{k+3}{6}$-subgaussian with parameter $b\sqrt{2\pi}$.*

Proof. First, we will show that x_{k-1} is a subgaussian and find the parameter for x_{k-1}. x_{k-1} has four possible value $\{a_0, a_0+1, a_1, a_1+1\}$. Since we use a uniform distribution, $\mathbb{E}[x_{k-1}] = \frac{1}{4}(a_0+a_0+1+a_1+a_1+1) = \frac{a_0+a_1+1}{2}$.

Algorithm 1 $\mathsf{Precompute}(b, q)$

```
Input: $q, b$
Output: $k = \lceil \log_b q \rceil$, $\mathbf{y} \in \{-1, 0\}^k$, $\mathbf{z} = \mathbf{S}\mathbf{y}$
$k = \lceil \log_b q \rceil$
for $i \leftarrow 0, \ldots, k-1$ do
    $y_i \xleftarrow{\$} \{-1, 0\}$
end for
$\mathbf{z} \leftarrow \mathbf{S}\mathbf{y}$
return $k, \mathbf{y}, \mathbf{z}$
```

Algorithm 2 $\mathsf{Subgaussian}'(b, q, k, \mathbf{y}, u)$: subgaussian sampling for $q = b^k$ of [JLP21]

```
Input: $u \in \mathbb{Z}_q$, $(k, \mathbf{y}, \mathbf{z}) \leftarrow \mathsf{Precompute}(b, q)$
Output: $\mathbf{x} \in \Lambda_u^{\perp}(\mathbf{g}^t)$ distributed uniformly in a bounded set.
Let $\mathbf{u} := [u]_b^k$ ($[u]_b^k$ is $u$'s $b$-ary decomposition)
$\mathbf{x} \leftarrow \mathbf{z} + \mathbf{u}$
return $\mathbf{x}$
```

Algorithm 3 $\mathsf{Subgaussian}(b, q, k, \mathbf{y}, u)$: our subgaussian sampling for $q < b^k$

```
Input: $u \in \mathbb{Z}_q$, $(k, \mathbf{y}, \mathbf{z}) \leftarrow \mathsf{Precompute}(b, q)$
Output: $\mathbf{x} \in \Lambda_u^{\perp}(\mathbf{g}^t)$ distributed uniformly in a bounded set.
if $y_{k-1} = 0$ then
    $u' \leftarrow u \mod b^{k-1}$
    $a \leftarrow \lfloor \frac{u}{b^{k-1}} \rfloor$
else
    $u' \leftarrow u - q \mod b^{k-1}$
    $a \leftarrow \lfloor \frac{u-q}{b^{k-1}} \rfloor$
end if
$\mathbf{x}' = \mathsf{Subgaussian}'(b, b^{k-1}, k-1, \mathbf{y}', u')$ where $\mathbf{y} = (\mathbf{y}', y_{k-1})$
if $y_{k-2} = 0$ then
    return $\mathbf{x} = (\mathbf{x}', a)$
else
    return $\mathbf{x} = (\mathbf{x}', a+1)$
end if
```

Let $\alpha = \mathbb{E}[x_{k-1}] = \frac{a_0+a_1+1}{2}$, then $|\alpha| \leq b - \frac{1}{2} \leq b$ since $-b \leq a_0, a_1 < b$. Let $\overline{x} = x_{k-1} - \alpha$, then $\overline{x}$ is a random variable centered at 0 (i.e., $\mathbb{E}[\overline{x}] = 0$) and $|\overline{x}| \leq \frac{b}{2}$. Hence $\overline{x}$ is 0-subgaussian with parameter $\overline{s} = \frac{b}{2}\sqrt{2\pi}$. By Lemma 1, if $s > \overline{s}$ and $\delta \geq \alpha^2\pi/(s^2 - \overline{s}^2)$, then x_{k-1} is δ-subgaussian with parameter s. Since

$$\frac{\alpha^2\pi}{s^2 - \overline{s}^2} \leq \frac{b^2\pi}{b^2 2\pi - (b^2/4)2\pi} = \frac{2}{3},$$

x_{k-1} is $\frac{2}{3}$-subgaussian with parameter $b\sqrt{2\pi}$.

Since $\mathbf{x}'$ is the output of $\mathsf{Subgaussian}'(b, b^{k-1}, k-1, \mathbf{y}, u)$, and in proof of Theorem 1 of [JLP21], $x_0, \ldots, x_{k-2}$ are $\frac{1}{6}$-subgaussian with parameter $b\sqrt{2\pi}$. Analogously to the proof of Lemma 4,

$$\begin{aligned}
\mathbb{E}[\exp(2\pi t\langle \mathbf{x}, \mathbf{u}\rangle)] &= \mathbb{E}[\exp(2\pi t \sum x_i u_i)] \\
&= \mathbb{E}[\exp(2\pi t \sum_{i=0}^{k-2} x_i u_i + 2\pi t x_{k-1} u_{k-1})] \\
&\leq \exp(\frac{k-1}{6} + \frac{2}{3}) \exp(\pi t^2 \max(s_i^2)) \\
&= \exp(\frac{k+3}{6}) \exp(\pi t^2 (b\sqrt{2\pi})^2).
\end{aligned}$$

Therefore, $\mathbf{x} = (\mathbf{x}', x_{k-1})$ is $\frac{k+3}{6}$-subgaussian with parameter $b\sqrt{2\pi}$.

□

4 Comparison with Previous Works

Experimental Setup All experiments are performed on a laptop with Apple M1 @ 3.2 GHz (8 cores). We used PALISADE Library [pal22] to implement our algorithm.

4.1 Randomness

`GMP19` uses $k \log q = O(k^2 \log b)$ random bits to sample an output with a certain probability. It is better to generate less number of random bits to achieve faster implementation result. `ZY22` has improved the number of random bits which is $O(k \log b)$ due to the randomness-efficient subroutine for a modulus b^{k-1}. Since our algorithm follows a uniform distribution over $\{-1, 0\}$, we only generate $k \log 2 = O(k)$ random bits. Therefore, we have faster implementation result than the others. However, in the offline phase when we sample $\mathbf{y}$, we store k random bits for being used in the online phase, hence we have additional small memory overhead.

4.2 Magnitude Comparison

Remark 1. Unlike other previous works employing centered distributions, the output of our algorithm has non-zero mean value. Therefore, we note that the noise analysis with our algorithm in homomorphic encryption is less simple than the one with centered distribution. In more detail, the (average-case) noise analysis of HE (especially GSW-like schemes) highly relies on the variance of the product of two independent polynomials x, y of degree N. The two random variable x and y are δ-subgaussian with parameter s. Then the variance of $x \cdot y$ is bounded as follows: $\mathsf{Var}(x \cdot y) \leq N \cdot \mathsf{Var}(x) \cdot \mathsf{Var}(y) + \mathbb{E}(x)^2 \cdot \mathsf{Var}(y) + \mathbb{E}(y)^2 \cdot \mathsf{Var}(x)$. If x and y have both zero mean, then it has clear and simple bound (the last two terms are eliminated). However, in our case, the mean of x and y are non-zero but less than b (since it is a digit decomposition with the base b). In GSW-like schemes [CGGI20, BIP$^+$22], for example, $N \gg b$ and $\mathsf{Var}(x) \cdot \mathsf{Var}(y)$ is the dominant term of the noise after homomorphic operation. After bootstrapping of TFHE, the final noise contained in the output has the variance of the sum of $\sum_{i \in [n]} \mathsf{Var}(x_i \cdot y_i)$, hence the dominant term is still unchanged, where x_i's and y_i's are independent subgaussian variables, where $|\mathbb{E}(x_i)| \leq b$ and $|\mathbb{E}(y_i)| = 0$ for all i. As a result, the bound is still tighter than the bound of worst-case with Euclidean/infinite norm.

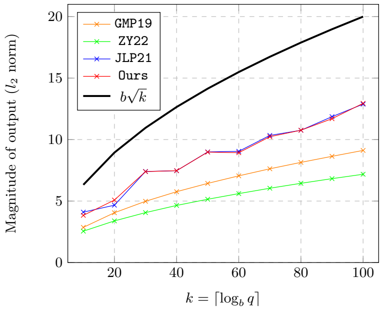

Fig. 1. the magnitude average of 10000 runs with $b = 2$, uniform random input u, and different moduli.

As we see from the remark above, the variance of the noise will have slightly larger bound if an FHE scheme uses a δ-subgaussian distribution, than the one using 0-subgaussian distribution, but the tighter bound is still preserved. We show the size of output in each case in the figure above.

Figure 1 shows the magnitude of output of each algorithm. We executed the experiment for 10000 runs with the base $b = 2$, and an uniform random input u by increasing the modulus q. In Section 3, we show that our algorithm outputs a δ-subgaussian vector with the parameter $\delta = \frac{k+3}{6}$ and the standard parameter $s = b\sqrt{2\pi}$.

Our standard parameter s is similar to other algorithms, and s is the upper bound of the standard deviation of the distribution. The variance of ours is also bounded by standard parameter $2b^2\pi$ by Lemma 2 like other non-uniform algorithms. Since our δ is nonzero unlike other sequential algorithms GMP19 and ZY22, the mean of the distribution is also nonzero. Therefore, ours is expected to have larger size of output than the others since ours has slighltly larger mean (still less than b).

The experimental result shows that the magnitude of the output is a little bit larger than them as we expected. But, in practice, we see that the output is much smaller than the least upper bound $b\sqrt{k}$ of Euclidean norm of outputs. Moreover, we have similar magnitude to JLP21 because we have the similar values of δ and s due to the use of the same uniform distribution. As discussed in [ZY22], GMP19 has slightly larger s than ZY22 with the same $\delta = 0$, hence they have smaller magnitude in general.

4.3 Complexity Comparison

In order to analyze the complexity of each subgaussian algorithm in detail, we divide each into its main subalgorithms and compute complexity of each subalgorithm in terms of the number of bit operations (see Table 1).

First, we briefly recall the subalgorithms of each algorithm. GMP19 consists of Decomposition, Transformation, Sampling, and Addition. Decomposition is the deterministic algorithm which outputs b-ary decomposition of u. Transformation computes $\mathbf{t} = \mathbf{S}^{-1}\mathbf{u}$ where $\mathbf{u}$ is the output of Decomposition to use $\mathbf{D}$. Sampling chooses a vector in $\Lambda_q^{\perp}(\mathbf{g}^t)$ with a subgaussian distribution centered at $-\mathbf{u}$. Addition combines the output vectors of Decomposition and Sampling to obtain a vector in $\Lambda_u^{\perp}(\mathbf{g}^t)$.

JLP21 consists of Sampling, Decomposition, and Addition. Here, Sampling, of which the complexity is $O(k)$, is done during offline due to a uniform distribution. Hence, it is not included in the time cost. Our algorithm also have the same algorithm, Algorithm 1, in offline phase. We divide ZY22 into four subalgorithms; Probability, Compute, Subgaussian′, and Check. The algorithm Probability computes the first probability $\frac{u}{q}$ to determine which u' and a are used. Compute computes u' and a which is same with the line 2-3 or 5-6 in Algorithm 3. Subgaussian′ randomly outputs a vector $\mathbf{x}'$ such that $\langle \mathbf{g}', \mathbf{x}' \rangle = u' \mod b^{k-1}$.

In fact, the algorithm Subgaussian′ is the power-of-base algorithm in [GMP19] which is efficient and easy to be implemented. Check checks whether $\langle \mathbf{g}', \mathbf{x}' \rangle = u'$ or $u' - b^{k-1}$. Ours follows the structure in [ZY22], thus we also have the same process as Compute of ZY22 does but our algorithm is much simpler. In addition, Subgaussian′ of our case is the power-of-base algorithm in [JLP21].

	Decomposition	Sampling	Transformation	Addition
GMP19	$O(k^2(\ell^2+\ell))$	$k \cdot P$	$k \cdot T$	$k(\ell^2+3\ell)$
JLP21	$O(k^2(\ell^2+\ell))$	N/A	N/A	$k(2\ell^2+5\ell)$
	Subgaussian′	Probability	Compute	Check
ZY22	$O(k^2(\ell^2+\ell)+k\ell)$	p	c_1	$O(k\ell)$
Ours	$O(k^2(\ell^2+\ell)+k\ell)$	N/A	c_2	N/A

Table 1. The number of bit operations of the subalgorithms of each sampler for fixed $q \approx b^k$, where $k = \lceil \log_b q \rceil$ and $\ell = \log_2 b$. T is the running time for computing each entry of $\mathbf{t}$. P denotes the computation time of sampling from given distribution. p denotes the computation time of computing the probability of ZY22. c_1, c_2 are for computing u' and a of the modulus reduction sampler. P, p, c_1, c_2 are constants which do not depend on k for given u and q.

Decomposition has the complexity $O(k^2(\ell^2+\ell))$ in terms of the number of bit operations since they compute each component of a k-dimensional vector using u and q whose the bit length is k times bit lengths of b (i.e., k times computation with $k\ell$ bit lengths numbers). Subgaussian′ runs Decomposition up to the $(k-1)$-th component of the output for the modulus b^{k-1}. Additionally, it contains the addition of two vectors of dimension of $k-1$.

On the other hand, Addition and Check require k times arithmetic operations over $\mathbb{Z}_b$, so that they depend on k and $\log b$. Transformation and Sampling also require computation of each component of k-dimensional vector, but of floating-point operations which consume constant cost. The operations are independent of a fixed q, hence, we denote the complexity of floating-point operations in Transformation and Sampling constant, denoted by T and P respectively, for convenience.

Similarly, the complexity of Probability, Compute of ZY22, and our Compute is denoted by constants p, c_1 and c_2 respectively, since both only compute $\frac{u}{q}, u'$ and a on floating-point numbers. Subgaussian′ of ZY22 consists of Decomposition, Sampling, and Addition which adds the output of Decomposition and Sampling of GMP19's power-of-base case, whereas the one of ours consists of Decomposition and Addition of JLP21's power-of-base case.

Overall, the dominant complexity comes from Decomposition , which is highly depends on the choice of parameters ℓ and k. Moreover, ZY22 and ours has constant factor in one of subalgorithms, in practice, the constant time can be a key factor which decides shortest running time in total.

4.4 Computation Cost Comparison

We compare the actual time cost of existing subgaussian decomposition algorithms [JLP21, ZY22] and ours. We do not include the experimental result of GMP19 for small modulus q since the comparison to GMP19 is already covered in [ZY22, JLP21]. For larger modulus $q \geq 2^{100}$, we included GMP19 as well since there has been no analysis about the algorithm with such q. We note that all our experiments consider the case that the modulus q is not a power of the base b (general case).

b	$k = \lceil \log_b q \rceil$	ZY22 [μs]	JLP21 [μs]	Ours [μs]
2^1	60	1.2709	0.5439	0.3539
2^2	30	0.6634	0.1888	0.1547
2^3	20	0.4723	0.1077	0.1008
2^4	15	0.3650	0.0762	0.0692
2^6	10	0.2628	0.0502	0.0477
2^8	8	0.1431	0.0441	0.0412

Table 2. Average runtimes for 10000 runs of subgaussian sampling for $\log_2 q \approx 60$ and uniformly random input $u \in \mathbb{Z}_q$ with the different base b.

	(b,k)	$(2,60)$	$(2^2,30)$	$(2^4,15)$	$(2^8,8)$
ZY22[μs]	Subgaussian$'$	1.2163	0.6152	0.3165	0.0959
	Compute	0.0192	0.0170	0.0167	0.0163
	Check	0.0180	0.0156	0.0152	0.0154
	Probability	0.0174	0.0156	0.0167	0.0154
JLP21[μs]	Decomposition	0.3355	0.1349	0.0553	0.0263
	Addition	0.2084	0.0540	0.0210	0.0178
Ours[μs]	Subgaussian$'$	0.3358	0.1383	0.0529	0.0253
	Compute	0.0181	0.0163	0.0163	0.0159

Table 3. Average runtimes of 10000 runs of each subalgorithm for $\log_2 q \approx 60$ and uniformly random input u with the different b(as a result with the different k).

For small modulus q As we already checked the complexity in Table 1, Decomposition has the dominant complexity, so it takes the dominant time in our experimental result. We can see that Decomposition of JLP21 and Subgaussian$'$ of both ZY22 and Ours takes the dominant time in the detailed time cost of Table 2 and Table 3. Despite of all the plausible tricks of ZY22, the computation time of JLP21 outperforms ZY22 due to the benefit of the precomputation.

Even though Subgaussian$'$ of Ours runs one less iterations of deterministic decomposition algorithm, it includes additional operations (we refer Algorithm 1 of [JLP21]), hence it takes slightly longer than Decomposition of JLP21. However, when k is large, Addition of JLP21 is significantly slower than our other subalgorithm Compute which is independent of k. That is why there is the biggest performance gap between JLP21 and Ours with the largest k, i.e., our algorithm is 35% faster than JLP21. As k decreases, the computation time of Addition becomes small as k decreases, even similar to Compute of Ours. Therefore, there becomes almost no difference between JLP21 and Ours as k significantly decreases.

The reason why Ours is faster than ZY22 is that our algorithm makes use of precomputed value as JLP21 does, hence the dominant part, Subgaussian$'$ of Ours, is faster than the one of ZY22. Moreover, Ours has faster Compute, and it does not need additional subalgorithms like Check and Probability. As a result, our algorithm is 77% $\sim$ 82% faster than ZY22.

Many applications of homomorphic encryption [PT20, CCR19, CDPP22, MCR21] use various value of k to achieve both the best performance and correctness. As we explained in Section 1, increasing k in subgaussian sampling causes lower noise growth resulting in supporting more homomorphic operations, but also slower performance at the same time. Therefore, the choice of parameter k is highly depends on the application. With our experimental result, we can conclude that for those applications which uses large k for $q \approx 2^{60}$, our algorithm is highly recommended.

b	$k = \lceil \log_b q \rceil$	GMP19[μs]	ZY22[μs]	JLP21[μs]	Ours[μs]
2^1	102	20.6274	22.3322	19.3688	19.0677
2^2	51	12.1608	13.5447	11.5216	11.3985
2^3	34	9.6771	10.7968	9.1633	9.1606
2^4	26	6.4006	7.2784	6.0475	6.0294
2^6	17	4.9778	5.6145	4.7116	4.7783
2^8	13	3.2420	3.6676	3.0062	3.0776

Table 4. Average runtimes for 10000 runs of subgaussian sampling for BigInteger q such that $\log_2 q \approx 102$ and uniformly random input u with the different base b.

	(b, k)	$(2, 102)$	$(2^2, 51)$	$(2^4, 26)$	$(2^8, 13)$
GMP19 [μs]	Decomposition	18.9774	11.3175	5.9694	2.9893
	Sampling	1.1783	0.6206	0.3133	0.1758
	Transformation	0.3864	0.1813	0.0923	0.0568
	Addition	0.0854	0.0413	0.0256	0.0200
ZY22 [μs]	Subgaussian′	22.0206	13.2311	6.9424	3.3370
	Compute	0.2743	0.2793	0.3024	0.2970
	Check	0.0161	0.0159	0.0155	0.0154
	Probability	0.0212	0.0184	0.0181	0.0182
JLP21 [μs]	Decomposition	18.9793	11.3154	5.9718	2.9814
	Addition	0.3896	0.2062	0.0757	0.0248
Ours [μs]	Subgaussian′	18.7783	11.0823	5.7161	2.7584
	Compute	0.2895	0.3162	0.3132	0.3192

Table 5. Average runtimes of 10000 runs of each subalgorithm for BigInteger q such that $\log_2 q \approx 102$ and uniformly random input u with the different b(as a result with the different k).

For large modulus q Zhang and Yu [ZY22] and Jeon et al. [JLP21] compared the running time of their algorithm and GMP19 only when $q \approx 2^{60}$. We provide the experimental result with larger $q \approx 2^{102}$, which shows that ZY22 is not always faster than GMP19 with such larger modulus.

The total running time increases significantly compared to the smaller q case. It is mainly because computation over numbers of BigInteger type, which was used in PALISADE library [pal22], takes more time than the other smaller bit length setting. Therefore, Decomposition which deals with large bit length has the dominant computation time, so do Subgaussian′ of ZY22 and Ours. However, Sampling of GMP19 deals with floating-point numbers which has the length less than 64-bits, so that there is a huge computation gap between Sampling and Decomposition of GMP19. Subgaussian′ of ZY22 contains Sampling and Decomposition of GMP19. But Sampling in their implementation is done over larger integer type BigInteger to compute probability, hence it takes longer time than GMP19. Therefore, ZY22 is slower than GMP19 with larger q in total when k is small.

Apart from ZY22, all the algorithms take similar time, it is because Sampling which samples a bit for each element takes negligible time comparing to Decomposition for large q, so the precomputation does not make any difference in this case. Interestingly, Compute of Ours takes more time than the case when q is small since it depends of q. Consequently, as mentioned above, there is a point that Addition of JLP21 becomes faster than Compute when k is small. This small difference makes JLP21 take the shortest time in total.

However, since all implementations of homomorphic encryption we are aware of use RNS technique to use 64-bit machine language for large ciphertext modulus in practice, the result of Table 2 is helpful for such cases.

b	k	Decomposition [μs]	ZY22 [μs]	JLP21 [μs]	Ours [μs]
2^1	60	640.7155	2418.6132	1050.4242	647.5024
2^2	30	311.9202	1346.4805	395.8116	315.4322
2^4	15	145.9810	695.3398	162.1303	148.0831
2^8	8	50.0841	381.8072	73.8022	57.6720

Table 6. Comparison of performance results between the deterministic decomposition and subgaussian samplings. The number of trial: 10000 for a 60-bits modulus q and n-dimensional input where $n = 2048$, with the different base b.

The gadget decomposition of homomorphic encryption takes n-dimensional vector $\mathbf{u}$ on input and outputs $\mathbf{x}$ such that $\mathbf{Gx} = \mathbf{u}$ for $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}^t$. We run the deterministic decomposition, which is same with

Decomposition, and subgaussian algorithms for a 60-bits modulus q and a 2048-dimensional input with the different base b (see Table 6). We note that these parameters are commonly used in many HE based applications [PT20, CCR19, CDPP22, ACLS18, CDNP23] to achieve high security level (larger than 110 bits of security). Obviously, the subgaussian algorithms take more time than the deterministic decomposition since they need more process to sample a random output.

We can check the computational overhead which shows how much the extra step costs than just running deterministic decomposition in order to see which one is suitable for the practical use. As we can see that from the table above, ZY22 has the largest computational overhead, whereas Ours has the least overhead in any choice of k. The overhead varies from 2% to 14% depending on k.

5 Conclusion

We propose a faster subgaussian decomposition by combining ZY22 and JLP21. To incorporate ZY22 and JLP21, we replace the call of GMP19 by the call of JLP21 in the structure of ZY22. And we also prove that a bounded uniform distribution is also subgaussian in the structure of ZY22. Previous works, GMP19, JLP21, and ZY22, also output actual subgaussian vectors so that they can be applied to HE schemes to analyze the noise growth without a Heuristic assumption. However, in the perspective of efficiency, they are too slower than the deterministic decomposition as shown in Table 6. In contrast, our algorithm has the lowest overhead only in the range from 2% to 14% to obtain actual subgaussian outputs. In addition, we give a detailed comparison, even for large modulus, with all the competitive algorithms, allowing applications to choose the best algorithm for their choice of parameters.

Acknowledgement

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2021R1A2C1094821). The third-listed author (J.Park) has been supported by CyberSecurity Research Flanders with reference number VR20192203.

References

ACLS18. Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society Press, May 2018.

AP14. Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314. Springer, Heidelberg, August 2014.

BdMW16. Florian Bourse, Rafaël del Pino, Michele Minelli, and Hoeteck Wee. FHE circuit privacy almost for free. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 62–89. Springer, Heidelberg, August 2016.

BGG$^+$14. Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

BIP$^+$22. Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. Final: Faster fhe instantiated with ntru and lwe. Cryptology ePrint Archive, Paper 2022/074, 2022. https://eprint.iacr.org/2022/074.

CCR19. Hao Chen, Ilaria Chillotti, and Ling Ren. Onion ring ORAM: Efficient constant bandwidth oblivious RAM from (leveled) TFHE. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 345–360. ACM Press, November 2019.

CDNP23. Kelong Cong, Debajyoti Das, Georgio Nicolas, and Jeongeun Park. Panacea: Non-interactive and stateless oblivious ram. Cryptology ePrint Archive, Paper 2023/274, 2023. https://eprint.iacr.org/2023/274.

CDPP22. Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V.L. Pereira. Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 563–577, New York, NY, USA, 2022. Association for Computing Machinery.

CGGI16. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2016.

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, January 2020.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Heidelberg, December 2017.

CM15. Sanjit Chatterjee and Alfred Menezes. Type 2 structure-preserving signature schemes revisited. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 286–310. Springer, Heidelberg, November / December 2015.

DDP$^+$18. W. Dai, Y. Doröz, Y. Polyakov, K. Rohloff, H. Sajjadpour, E. Savaş, and B. Sunar. Implementation and evaluation of a lattice-based key-policy abe scheme. *IEEE Transactions on Information Forensics and Security*, 13(5):1169–1184, 2018.

DM15. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Heidelberg, April 2015.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

GM18. Nicholas Genise and Daniele Micciancio. Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 174–203. Springer, Heidelberg, April / May 2018.

GMP19. Nicholas Genise, Daniele Micciancio, and Yuriy Polyakov. Building an efficient lattice gadget toolkit: Subgaussian sampling and more. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 655–684. Springer, Heidelberg, May 2019.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

JLP21. Sohyun Jeon, Hyang-Sook Lee, and Jeongeun Park. Efficient lattice gadget decomposition algorithm with bounded uniform distribution. *IEEE Access*, 9:17429–17437, 2021.

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Heidelberg, May 2013.

MCR21. Muhammad Haris Mughees, Hao Chen, and Ling Ren. OnionPIR: Response efficient single-server PIR. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2292–2306. ACM Press, November 2021.

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

MP19. Sean Murphy and Rachel Player. δ-subgaussian random variables in cryptography. In *Information Security and Privacy*, pages 251–268. Springer, 2019.

pal22. PALISADE Lattice Cryptography Library (release 1.11.6). https://palisade-crypto.org/, January 2022.

PT20. Jeongeun Park and Mehdi Tibouchi. SHECS-PIR: Somewhat homomorphic encryption-based compact and scalable private information retrieval. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 86–106. Springer, Heidelberg, September 2020.

Str94. K. Stromberg. *Probability For Analysts.* Chapman & Hall/CRC Probability Series. Taylor & Francis, 1994.

ZY22. Shiduo Zhang and Yang Yu. Towards a simpler lattice gadget toolkit. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography – PKC 2022*, pages 498–520, Cham, 2022. Springer International Publishing.