Towards a Simpler Lattice Gadget Toolkit

Shiduo Zhang[1] and Yang Yu[2⋆]

Institute for Advanced Study, Tsinghua University, Beijing, China
BNRist, Tsinghua University, Beijing, China
yang.yu0986@gmail.com

Abstract. As a building block, gadgets and associated algorithms are widely used in advanced lattice cryptosystems. The gadget algorithms for power-of-base moduli are very efficient and simple, however the current algorithms for arbitrary moduli are still complicated and practically more costly despite several efforts. Considering the necessity of arbitrary moduli, developing simpler and more practical gadget algorithms for arbitrary moduli is crucial to improving the practical performance of lattice based applications.

In this work, we propose two new gadget sampling algorithms for arbitrary moduli. Our first algorithm is for gadget Gaussian sampling. It is simple and efficient. One distinguishing feature of our Gaussian sampler is that it does not need floating-point arithmetic, which makes it better compatible with constrained environments. Our second algorithm is for gadget subgaussian sampling. Compared with the existing algorithm, it is simpler, faster, and requires asymptotically less randomness. In addition, our subgaussian sampler achieves an almost equal quality for different practical parameters. Overall these two algorithms provide simpler options for gadget algorithms and enhance the practicality of the gadget toolkit.

1 Introduction

Lattice based cryptography is not only a strong contender in the NIST post-quantum standardization, but also offers powerful versatility leading to the constructions of various advanced cryptographic primitives ranging from identity based encryption (IBE) [13, 1], attribute based encryption (ABE) [27], group signatures [29, 34, 36] to fully homomorphic encryption (FHE) [24, 26, 15], functional encryption [3, 2, 35] and much more [7, 28]. Many advanced lattice cryptosystems rely on strong *lattice trapdoors* that allow to sample lattice points from Gaussian-like distributions. The notion of lattice trapdoor was introduced in [25] along with a sampling algorithm. Later a series of works [37, 19, 20, 14, 18] proposed improved trapdoor constructions and sampling algorithms.

Currently, the state-of-the-art lattice trapdoor framework is developed by Micciancio and Peikert [37]. Following the idea of [39], the trapdoor sampling in this framework is decomposed into online and offline two phases, and the online

⋆ Corresponding Author.

sampling is accomplished by the sampling over a special lattice $\Lambda^{\perp}(\mathbf{g}^t) = \{\mathbf{z} \in \mathbb{Z}^k : \mathbf{g}^t\mathbf{z} = 0 \mod q\}$ defined by the *gadget* $\mathbf{g} = (1, b, \cdots, b^{k-1})$. Thanks to the good structure of the short basis of $\Lambda^{\perp}(\mathbf{g}^t)$, the sampling over $\Lambda^{\perp}(\mathbf{g}^t)$ is convenient and fast, which improves the efficiency of the online sampling. As a building block of lattice based cryptography, gadgets have been in effect used in much more applications, e.g. [26, 11, 12]. In summary, the use of the gadget is mainly based on four algorithms:

- **Digit Decomposition:** Given $u \in \mathbb{Z}_q$, find a short $\mathbf{x}$ such that $\langle \mathbf{x}, \mathbf{g} \rangle = u \bmod q$. This is the most widely used case, identifying a number of size $O(b^k)$ with a vector of norm $O(b\sqrt{k})$.
- **LWE Decoding:** Given $s\mathbf{g} + \mathbf{e} \bmod q$ for a sufficiently small $\mathbf{e}$, recover s. This algorithm is deterministic as the digit decomposition and a representative usecase is in the decryption of LWE cryptosystems.
- **Gaussian Sampling:** Given $u \in \mathbb{Z}_q$, sample $\mathbf{x}$ from a discrete Gaussian on a lattice coset $\Lambda_u^{\perp}(\mathbf{g}^t) = \{\mathbf{z} \in \mathbb{Z}^k : \mathbf{g}^t\mathbf{z} = u \bmod q\}$. This algorithm is randomized unlike the digit decomposition and LWE decoding. It is the main component of the Micciancio-Peikert trapdoor [37] and used in lattice based signatures, IBE and many other primitives.
- **Subgaussian Sampling:** Given $u \in \mathbb{Z}_q$, sample a subgaussian $\mathbf{x}$ in $\Lambda_u^{\perp}(\mathbf{g}^t)$. This algorithm is also randomized and can work with much less randomness than the Gaussian sampling. It is used in some FHE schemes as an alternative to the digit decomposition for tighter parameters [4].

Micciancio and Peikert gave very efficient gadget Gaussian sampling and LWE decoding algorithms in [37] but mainly for the special case where the modulus $q = b^k$. Genise and Micciancio later proposed an equally efficient (in an asymptotic sense) gadget Gaussian sampler for arbitrary moduli [21]. Genise, Micciancio and Polyakov also devised gadget subgaussian sampling and LWE decoding algorithms applicable to an arbitrary modulus q [23]. With these efforts, recent years have seen significant progress in bringing advanced lattice cryptosystems in practice [17, 16, 30, 9, 8].

Despite the same asymptotic complexity, there still exist some gaps between the practicalities of the gadget algorithms for the special $q = b^k$ and for an arbitrary modulus $q < b^k$. The specialized algorithms for $q = b^k$ are very simple and only require integer operations. In contrast, the existing algorithms for $q < b^k$ are complicated, and particularly the Gaussian sampler has to resort to high-precision arithmetic, which limits the use of gadget algorithms on some constrained devices. To close these gaps is not only of theoretical interest but also crucial for practical applications: many lattice cryptosystems require the modulus q to support the NTT/RNS/CRT techniques for better performance, hence $q < b^k$ in these cases.

Our contribution. Towards better practicality of the gadget toolkit, we improve on two randomized gadget algorithms, i.e. Gaussian sampling and subgaussian sampling, for arbitrary moduli.

We present a new gadget Gaussian sampler that avoids the floating-point arithmetic in existing algorithms. Compared with the previous algorithms, our sampler achieves the same quality and asymptotic complexity, but is simpler and highly parallelizable. Verified by experiments (see Figures 1 and 2), our sampler is as fast as the original Genise-Micciancio sampler [21] for practical parameters, but slower than an improved variant of the Genise-Micciancio sampler [16] in which continuous Gaussian sampling is heavily used.

We also propose a new gadget subgaussian sampler. It does not use any linear transformation and most computations are identical to those in the specialized algorithm for $q = b^k$. Consequently, the new sampler is simpler, faster and only requires $O(k \log b)$-bits of randomness, which improves the previous (considered essentially optimal) result by $O(k)$. Indeed the subgaussian parameter achieved by our algorithm may be $\sqrt{2}$ times as large as that by the previous algorithm in the worst case. But it is convenient to get an almost equal quality in practice by selecting proper parameters without speed and security loss.

In summary, we provide the gadget toolkit with simpler algorithmic options. Due to the absence of high-precision arithmetic, the new gadget Gaussian sampler is of some interest when side-channel protections and constrained devices are taken into account. The new subgaussian sampler can be used to improve the efficiency and simplicity of the implementation of advanced lattice schemes.

Techniques. We now briefly explain the used techniques. In this work, we focus on the gadget $\mathbf{g} = (1, b, \cdots, b^{k-1})$ and the gadget lattice $\Lambda^{\perp}(\mathbf{g}^t) = \{\mathbf{z} \in \mathbb{Z}^k : \mathbf{g}^t\mathbf{z} = 0 \mod q\}$.

Our gadget Gaussian sampling algorithm follows Peikert's approach [39]: it first generates a perturbation vector of certain covariance and then generates a Gaussian sample from an easy-to-sample lattice. Concretely, we represent the basis $\mathbf{B}_q$ of $\Lambda^{\perp}(\mathbf{g}^t)$ as $\mathbf{B}_q = \mathbf{TD}$ where $\mathbf{T}$, $\mathbf{D}$ were first suggested in [21]. With such a factorization, the sampling over $\Lambda^{\perp}(\mathbf{g}^t)$ is decomposed into the sampling over $\mathcal{L}(\mathbf{D})$ that is easy and over integers and the perturbation sampling of covariance $\mathbf{\Sigma} = s^2\mathbf{I} - \mathbf{TT}^t$ that introduces floating-point arithmetic. To avoid floating-point arithmetic, we exploit an integral matrix decomposition $\mathbf{\Sigma} = \mathbf{AJA}^t$ with $\mathbf{A} \in \mathbb{Z}^{k \times k'}$ and $\mathbf{J}$ being diagonal, which is inspired by [18]. But a technical difference is that the middle matrix $\mathbf{J}$ is a diagonal but not identity matrix, which allows to reduce the size of $\mathbf{A}$, that is only $k \times (k+2)$ much smaller than the size of the Gram root given in [18], while keeping $\mathbf{A}$ integral. With such a compact integral decomposition, the perturbation sampling can be done by applying a linear transformation of $\mathbf{A}$ on $D_{\mathbb{Z}^{k'}, \sqrt{\mathbf{J}}}$, which is simple, fast and highly parallelizable.

Our gadget subgaussian algorithm is very different from the one proposed by Genise, Micciancio and Polyakov [23]. The Genise-Micciancio-Polyakov algorithm relies on the factorization $\mathbf{B}_q = \mathbf{TD}$ used in the previous gadget Gaussian sampler [21]: it first performs subgaussian sampling with respect to $\mathbf{D}$ and then multiplies by $\mathbf{T}$ to get the final result. The subgaussian sampling with respect to $\mathbf{D}$ requires $O(k^2 \log b)$-bits of randomness while the specialized algorithm sampling directly over $\mathbf{B}_{b^k}$ needs only $O(k \log b)$-bits. The linear transformation $\mathbf{T}$

also introduces extra computational overhead. Consequently, a performance gap occurs between the Genise-Micciancio-Polyakov algorithm for $q = b^k$ and $q < b^k$. To close this gap, our idea is to convert the sampling for $q < b^k$ into the sampling for a power-of-b modulus. In a nutshell, we propose to first sample the $(k-1)$ lower digits by calling the subgaussian algorithm for the modulus b^{k-1} and then to compute the highest digit. Specifically, we sample within two sets $S_0 = \{\mathbf{x} \mid \langle \mathbf{x}, \mathbf{g} \rangle = u\}$ and $S_1 = \{\mathbf{x} \mid \langle \mathbf{x}, \mathbf{g} \rangle = u - q\}$ and in each set the highest digit x_{k-1} is basically fixed and so is $\langle \mathbf{x}', \mathbf{g}' \rangle \bmod b^{k-1}$ where $\mathbf{x}' = (x_0, \cdots, x_{k-2})$ and $\mathbf{g}' = (1, b, \cdots, b^{k-2})$. Our sampler proceeds in three steps. First, it chooses either S_0 or S_1 to which the output belongs. Once the set is fixed, it then calls the specialized algorithm to output a subgaussian $\mathbf{x}'$ given $\langle \mathbf{x}', \mathbf{g}' \rangle \bmod b^{k-1}$. Finally it computes x_{k-1} according to the chosen S_i and the exact value of $\langle \mathbf{x}', \mathbf{g}' \rangle$. To ensure $\mathbf{x}$ is subgaussian, it suffices to show the expectation of the output $\mathbf{x}$ is $\mathbf{0}$. To this end, we figure out the probability of S_i should be chosen according to u. As a consequence, we prove the subgaussian parameter of the output is at most $\sqrt{(b-1)^2 + \alpha^2}\sqrt{2\pi}$ with $\alpha = \lfloor q/b^{k-1} \rfloor + 1$, which can be close to even better than the previous result $(b+1)\sqrt{2\pi}$ for some practical (q, b). Thanks to the ease of the specialized algorithm, our subgaussian algorithm achieves better practical performance (see Figure 3) and requires $O(k \log b)$ random bits which asymptotically improves the previous result and is essentially identical to the case $q = b^k$.

Roadmap. We start in Section 2 with some preliminary material. Section 3 is devoted to recalling the state of the art of the gadget Gaussian and subgaussian samplers. We present our new gadget Gaussian and subgaussian sampling algorithms in Section 4 and Section 5 respectively. Finally, we conclude in Section 6.

2 Preliminaries

2.1 Notation

A number is denoted by a lower case letter, e.g. $z \in \mathbb{Z}$. A vector is denoted by a bold lower case letter, e.g. $\mathbf{v}$, and in column form ($\mathbf{v}^t$ is a row vector). The inner product of two vectors is $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^t \mathbf{y}$. Let $\mathbb{Z}_q = \{0, 1, \cdots, q-1\}$ for a positive integer q. For integers $b > 0$ and $u < b^k$, the b-ary decomposition of u is $[u]_b^k = (u_0, \cdots, u_{k-1}) \in \mathbb{Z}_b^k$ such that $\sum_i b^i u_i = u$. We denote matrices with bold upper case letters, e.g. $\mathbf{B}$. Let $\mathbf{B}^t$ be the transpose of $\mathbf{B}$. Unless otherwise stated, the norm of a vector is the ℓ_2 norm. Let $\|\mathbf{B}\|_{col} = max_i \|\mathbf{b}_i\|$. We use log and ln to denote the base 2 logarithm and the natural logarithm respectively. Let $\epsilon > 0$ be some very small number. We use the notational shortcut $\hat{\epsilon} = \epsilon + O(\epsilon^2)$. Then $\frac{1+\epsilon}{1-\epsilon} = 1 + 2\hat{\epsilon}$ and $\ln(\frac{1+\epsilon}{1-\epsilon}) = 2\hat{\epsilon}$.

A random variable x sampled from a distribution D is written as $x \leftarrow D$. A random variable distributed as D is denoted $x \sim D$. The *max-log distance* between two distributions D_1 and D_2 over the same support S is

$$\Delta_{ML}(D_1, D_2) = \max_{x \in S} |\ln(D_1(x)) - \ln(D_2(x))|.$$

As shown in [38], $\Delta_{ML}(D_1, D_2) \leq \Delta_{ML}(D_1, D_3) + \Delta_{ML}(D_2, D_3)$.

2.2 Linear Algebra

For $\mathbf{T} \in \mathbb{R}^{n \times k}$, let $\mathrm{span}(\mathbf{T})$ be the linear span of the columns of $\mathbf{T}$ and $\mathsf{ker}(\mathbf{T})$ be the kernel of $\mathbf{T}$. The (foreward) *Gram-Schmidt orthogonalization* of an ordered set of linearly independent vectors $\mathbf{B} = \{\mathbf{b}_1, \cdots, \mathbf{b}_k\}$ is $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, \cdots, \tilde{\mathbf{b}}_k\}$ where each $\tilde{\mathbf{b}}_i$ is the component of $\mathbf{b}_i$ orthogonal to $\mathrm{span}(\mathbf{b}_1, \cdots, \mathbf{b}_{i-1})$.

We write $\mathbf{\Sigma} > 0$ (resp., $\mathbf{\Sigma} \geq 0$) when a symmetric matrix $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ is *positive definite* (resp. *semidefinite*), i.e. $\mathbf{x}^t \mathbf{\Sigma} \mathbf{x} > 0$ (resp., $\mathbf{x}^t \mathbf{\Sigma} \mathbf{x} \geq 0$) for all nonzero $\mathbf{x} \in \mathbb{R}^n$. We write $\mathbf{\Sigma}_1 \geq \mathbf{\Sigma}_2$ or $\mathbf{\Sigma}_2 \leq \mathbf{\Sigma}_1$ if $\mathbf{\Sigma}_1 - \mathbf{\Sigma}_2 \geq 0$, and similarly for $\mathbf{\Sigma}_1 > \mathbf{\Sigma}_2$. It holds that $\mathbf{\Sigma}_1 > \mathbf{\Sigma}_2 > 0$ if and only if $\mathbf{\Sigma}_2^{-1} > \mathbf{\Sigma}_1^{-1} > 0$. If $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^t$, we call $\mathbf{A}$ a *Gram root* of $\mathbf{\Sigma}$. Let $\sqrt{\mathbf{\Sigma}}$ denote any Gram root of $\mathbf{\Sigma}$ when the context permits it.

2.3 Lattices

A lattice is the set of all integer combinations of linearly independent vectors $\mathbf{b}_1, \cdots, \mathbf{b}_n \in \mathbb{R}^m$. We call $\mathbf{B} = (\mathbf{b}_1, \cdots, \mathbf{b}_n)$ a basis and n the dimension of the lattice. The lattice is *full-rank* if $n = m$. We denote by $\mathcal{L}(\mathbf{B})$ the lattice generated by the basis $\mathbf{B}$. A coset of a lattice Λ is a set of the form $\{\mathbf{v} + \mathbf{a} | \mathbf{v} \in \Lambda\} := \Lambda + \mathbf{a}$. Let $\Lambda^* = \{\mathbf{x} \in \mathrm{span}(\Lambda) | \langle \mathbf{x}, \Lambda \rangle \subseteq \mathbb{Z}\}$ be the dual lattice of Λ.

2.4 Gaussian

The n-dimensional *Gaussian* function $\rho : \mathbb{R}^n \to (0, 1]$ is defined as $\rho(\mathbf{x}) := \exp(-\pi \|\mathbf{x}\|^2)$. Let $\rho_{\mathbf{B}}(\mathbf{x}) = \exp(-\pi \mathbf{x}^t \mathbf{\Sigma}^{-1} \mathbf{x})$ where $\mathbf{\Sigma} = \mathbf{B}\mathbf{B}^t$. Since $\rho_{\mathbf{B}}(\mathbf{x})$ is completely determined by $\mathbf{\Sigma} = \mathbf{B}\mathbf{B}^t$, we also write $\rho_{\sqrt{\mathbf{\Sigma}}}(\mathbf{x}) = \rho_{\mathbf{B}}(\mathbf{x})$. Let $\rho_{\sqrt{\mathbf{\Sigma}}, \mathbf{c}}(\mathbf{x}) = \rho_{\sqrt{\mathbf{\Sigma}}}(\mathbf{x} - \mathbf{c})$ for $\mathbf{c} \in \mathrm{span}(\mathbf{\Sigma})$. When $\mathbf{c} = \mathbf{0}$, we omit the subscript $\mathbf{c}$. For a countable set of $S \subset \mathbb{R}^n$, let $\rho_{\sqrt{\mathbf{\Sigma}}}(S) = \sum_{\mathbf{s} \in S} \rho_{\sqrt{\mathbf{\Sigma}}}(\mathbf{s})$. The *discrete Gaussian* over a lattice Λ with center $\mathbf{c}$ and covariance matrix $\mathbf{\Sigma}$ is defined by the probability function

$$D_{\Lambda, \sqrt{\mathbf{\Sigma}}, \mathbf{c}}(\mathbf{x}) = \frac{\rho_{\sqrt{\mathbf{\Sigma}}, \mathbf{c}}(\mathbf{x})}{\rho_{\sqrt{\mathbf{\Sigma}}, \mathbf{c}}(\Lambda)} \propto \rho_{\sqrt{\mathbf{\Sigma}}, \mathbf{c}}(\mathbf{x}).$$

The discrete Gaussian on $\Lambda + \mathbf{c}$, for $\mathbf{c} \in \mathrm{span}(\Lambda)$, is defined by $D_{\Lambda + \mathbf{c}, \sqrt{\mathbf{\Sigma}}}(\mathbf{x}) = \rho_{\sqrt{\mathbf{\Sigma}}}(\mathbf{x}) / \rho_{\sqrt{\mathbf{\Sigma}}}(\Lambda + \mathbf{c})$ for all $\mathbf{x} \in \Lambda + \mathbf{c}$. When $\mathbf{\Sigma} = s^2 \mathbf{I}$, we call the Gaussian *spherical* of *width* s and write the subscript $\sqrt{\mathbf{\Sigma}}$ as s simply.

For a lattice Λ and $\epsilon > 0$, $\eta_\epsilon(\Lambda) = \min\{s > 0 \mid \rho_{\frac{1}{s}}(\Lambda^*) \leq 1 + \epsilon\}$ is called the *smoothing parameter*. The following definition is a generalized version.

Definition 1 ([39], Definition 2.3). *Let $\mathbf{\Sigma} > 0$ and lattice $\Lambda \in \mathrm{span}(\mathbf{\Sigma})$. We write $\sqrt{\mathbf{\Sigma}} \geq \eta_\epsilon(\Lambda)$ if $\eta_\epsilon(\sqrt{\mathbf{\Sigma}}^{-1} \cdot \Lambda) \leq 1$ i.e. $\eta_{\sqrt{\mathbf{\Sigma}^{-1}}}(\Lambda^*) \leq 1 + \epsilon$.*

Notice that for two lattices of the same rank $\Lambda_1 \subseteq \Lambda_2$, the denser lattice always has the smaller smoothing parameter, i.e. $\eta_\epsilon(\Lambda_2) \leq \eta_\epsilon(\Lambda_1)$. Let $\overline{\eta_\epsilon}(\mathbb{Z}^n) = \sqrt{\frac{\ln(2n(1+1/\epsilon))}{\pi}}$. Here we recall several facts to be used later.

Lemma 1 ([25], Lemma 3.1). *Let $\Lambda \subset \mathbb{R}^n$ be a lattice with a basis $\mathbf{B}$, then $\eta_\epsilon(\Lambda) \leq \|\tilde{\mathbf{B}}\|_{col} \cdot \overline{\eta_\epsilon}(\mathbb{Z}^n)$.*

Theorem 1 (Adapted from Theorem 3.1 [22]). *For any $\epsilon \in [0,1)$, matrix $\mathbf{S}$ of full column rank, lattice $\Lambda \subset \mathrm{span}(\mathbf{S})$, and matrix $\mathbf{T}$ such that $\mathsf{ker}(\mathbf{T})$ is a Λ-subspace and $\eta_\epsilon(\Lambda \cap \mathsf{ker}(\mathbf{T})) \leq \mathbf{S}$, then $\Delta_{ML}(\mathbf{T} \cdot D_{\Lambda,\mathbf{S}}, D_{\mathbf{T}\Lambda,\mathbf{TS}}) \leq 2\hat{\epsilon}$.*

Theorem 2 (Adapted from Theorem 3.1 [39]). *Let $\mathbf{\Sigma}_1, \mathbf{\Sigma}_2 \in \mathbb{R}^{n\times n}$ be positive definite matrices. Let $\mathbf{\Sigma} = \mathbf{\Sigma}_1 + \mathbf{\Sigma}_2$ and let $\mathbf{\Sigma}_3 \in \mathbb{R}^{n\times n}$ be such that $\mathbf{\Sigma}_3^{-1} = \mathbf{\Sigma}_1^{-1} + \mathbf{\Sigma}_2^{-1}$. Let Λ_1, Λ_2 be two full-rank lattices in $\mathbb{R}^n$ such that $\sqrt{\mathbf{\Sigma}_1} \geq \eta_\epsilon(\Lambda_1)$ and $\sqrt{\mathbf{\Sigma}_3} \geq \eta_\epsilon(\Lambda_2)$ for $\epsilon \in (0, 1/2)$. Let $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{R}^n$, then the distribution of $\mathbf{x} \leftarrow D_{\Lambda_1,\sqrt{\mathbf{\Sigma}_1},\mathbf{p}-\mathbf{c}_2+\mathbf{c}_1}$ where $\mathbf{p} \leftarrow D_{\Lambda_2,\sqrt{\mathbf{\Sigma}_2},\mathbf{c}_2}$ is within max-log distance $4\hat{\epsilon}$ of $D_{\Lambda_1,\sqrt{\mathbf{\Sigma}},\mathbf{c}_1}$.*

2.5 Subgaussian Random Variables

A random variable X over $\mathbb{R}$ is *subgaussian* with parameter $\alpha > 0$ if for all $t \in \mathbb{R}$, its (scaled) moment generating function satisfies

$$\mathbb{E}[\exp(2\pi t X)] \leq \exp(\pi\alpha^2 t^2).$$

Scaling a subgaussian X with parameter α by any $c \in \mathbb{R}$ to $c \cdot X$ yields a subgaussian random variable with parameter $|c|\alpha$. If X is subgaussian with parameter α, then $\Pr[|X| \geq t] \leq 2\exp(-\pi t^2/\alpha^2)$. If X is a random variable with $\mathbb{E}(X) = 0$ and $|X| \leq b$ for some $b > 0$, then X is subgaussian with parameter $b\sqrt{2\pi}$ [40]. Moreover, if X is subgaussian variable, then $\mathbb{E}[X] = 0$. An important property of suggaussian called *Pythagorean additivity* is defined as follow.

Lemma 2. *Let X, Y be discrete random variables over $\mathbb{R}$ such that X is subgaussian with parameter α and Y conditioned on X taking any value is subgaussian with parameter β. Then, $X + Y$ is subgaussian with parameter $\sqrt{\alpha^2 + \beta^2}$.*

A random vector $\mathbf{x}$ over $\mathbb{R}^n$ is subgaussian with parameter $\alpha > 0$ if $\langle \mathbf{x}, \mathbf{u} \rangle$ is subgaussian with parameter α for all unit vectors $\mathbf{u}$.

Lemma 3. *Let $\mathbf{x}$ be a discrete random vector over $\mathbb{R}^n$ such that each coordinate x_i is subgaussian with parameter α_i given the previous coordinates take any values. Then, $\mathbf{x}$ is a subgaussian vector with parameter $\max_i\{\alpha_i\}$.*

3 Recall the Gadget Sampling

The gadget Gaussian and subgaussian samplings are two primary algorithms associated to the lattice gadget. For better completeness and comparisons, let us briefly recall the state of the art of these two algorithms.

Throughout the paper, we focus on the most widely used gadget defined by $\mathbf{g} = (1, b, \cdots, b^{k-1})$ where $b \in \mathbb{N}$ such that the global modulus $q \leq b^k$. The lattice $\Lambda^\perp(\mathbf{g}^t) = \{\mathbf{z} \in \mathbb{Z}^k : \mathbf{g}^t\mathbf{z} = 0 \bmod q\}$ is called the gadget lattice.

3.1 Gadget Gaussian Sampling

The goal of gadget Gaussian sampling is to generate a sample from a discrete Gaussian on a lattice coset $\Lambda_u^{\perp}(\mathbf{g}^t) = \{\mathbf{z} \in \mathbb{Z}^k : \mathbf{g}^t\mathbf{z} = u \bmod q\}$. The associated algorithms were developed by Micciancio and Peikert [37] to construct an efficient and powerful lattice trapdoor framework. As shown in [37], the gadget Gaussian sampling is convenient thanks to a good basis of $\Lambda^{\perp}(\mathbf{g}^t)$ as follows:

$$\mathbf{B}_q = \begin{pmatrix} b & & & q_0 \\ -1 & b & & q_1 \\ & -1 & \ddots & \vdots \\ & & \ddots & b \;\; q_{k-2} \\ & & & -1 \;\; q_{k-1} \end{pmatrix} \tag{1}$$

where $q = \sum_{i=0}^{k-1} b^i q_i$. Particularly, when $q = b^k$, $\mathbf{B}_q$ is bi-diagonal and thus its Gram-Schmidt orthogonalization in reverse order is diagonal, which leads to a very simple sampler that runs in $O(k)$ and is implemented over integers. But the sampler for $q < b^k$ proposed in [37] requires $O(k^2)$ complexity even with pre-computation.

Later, Genise and Micciancio proposed in [21] an improved gadget Gaussian sampler for $q < b^k$ that achieves the complexity of $O(k)$ as well. Their approach is build upon the Gaussian convolution technique [39]. In more details, they noticed a factorization $\mathbf{B}_q = \mathbf{TD}$ with

$$\mathbf{T} = \begin{pmatrix} b & & & \\ -1 & b & & \\ & -1 & \ddots & \\ & & \ddots & b \\ & & & -1 \; b \end{pmatrix} \quad \text{and} \quad \mathbf{D} = \begin{pmatrix} 1 & & & d_0 \\ & 1 & & d_1 \\ & & \ddots & \vdots \\ & & & 1 \; d_{k-2} \\ & & & d_{k-1} \end{pmatrix} \tag{2}$$

and then decomposed the sampling into two steps: generating $\mathbf{p} \leftarrow D_{\mathcal{L},r\sqrt{\mathbf{\Sigma}_2}}$ and outputting $\mathbf{T} \cdot D_{\mathcal{L}(\mathbf{D}),r,-\mathbf{c}}$ with $\mathbf{\Sigma}_2 = (s/r)^2\mathbf{I} - \mathbf{TT}^t$, $\mathbf{c} = \mathbf{T}^{-1}(\mathbf{u} - \mathbf{p})$. Originally, [21] proposed to set $\mathcal{L} = \mathcal{L}(\mathbf{\Sigma}_2)$ to simplify the sampling of $\mathbf{p}$. Later, [16] suggested to sample $\mathbf{p}$ via continuous Gaussian sampling, which turns out practically efficient given some high-precision arithmetic library.

Hu and Jia also gave a gadget Gaussian sampler for $q < b^k$ [32]: instead of sampling a spherical Gaussian, they proposed to sample a non-spherical one over $\Lambda^{\perp}(\mathbf{g}^t)$. This improves the efficiency of the gadget Gaussian sampling at the cost of complicating the offline sampling in the Micciancio-Peikert framework, which defeats some optimization techniques [21, 18]. In this paper, we are interested in spherical gadget Gaussian sampling as in [37, 21] that gives a better overall performance.

3.2 Gadget Subgaussian Sampling

The gadget subgaussian sampling produces a subgaussian vector on a lattice coset $\Lambda_u^\perp(\mathbf{g}^t)$. It has advantages over digit decomposition and gadget Gaussian sampling: the subgaussian has a "pythagorean additivity" property, which gives rise to tighter parameters than digit decomposition; the subgaussian sampling is faster and requires less randomness than the Gaussian one.

Genise, Micciancio and Polyakov initiated the study of gadget subgaussian sampling [23] and proposed two algorithms for $q = b^k$ and $q < b^k$ respectively. When $q = b^k$, the algorithm is in effect a specialized version of the Babai's nearest plane algorithm [5] on $\mathbf{B}_q$ (Eq. (1)). This algorithm achieves subgaussian parameter at most $(b-1)\sqrt{2\pi}$ with $O(k)$ operations and $\log q$ random bits.

The gadget subgaussian algorithm for arbitrary moduli, i.e. $q < b^k$, proceeds in a rather different manner. It exploits the same matrix factorization $\mathbf{B}_q = \mathbf{TD}$ (Eq. (2)) as in [21]. Concretely, it performs a specialized Babai's nearest plane algorithm on $\mathbf{D}$ and applies a linear transformation of $\mathbf{T}$ to lift the solution to $\Lambda_u^\perp(\mathbf{g}^t)$. In the end, this algorithm runs in linear $O(k)$ time, requires $O(k \log q)$ random bits and achieves subgaussian parameter at most $(b+1)\sqrt{2\pi}$. Overall the subgaussian algorithm for arbitrary moduli is more complicated and randomness inefficient than the one for $q = b^k$.

4 Gadget Gaussian Sampling without Floats

In contrast to the specialized gadget Gaussian sampler for $q = b^k$, the state-of-the-art sampler for arbitrary moduli is still complicated and heavily uses floating point arithmetic, despite the asymptotically same complexity (See Section 3.1). Floating point arithmetic has many drawbacks in practice in terms of security, numerical stability and efficiency. Particularly, once the gadget sampler relies on floating point operations, it would be inconvenient and inefficient to deploy the trapdoor cryptosystems [37] in constraint devices[1].

Here we present a new gadget Gaussian sampler for arbitrary moduli but without using floating point arithmetic. Our sampler achieves the complexity of $O(k)$ as the Genise-Micciancio sampler [21] and is even simpler. Moreover, the practical running time of our sampler is close to that of the Genise-Micciancio sampler; a large part of samplings in our algorithm are parallelizable.

[1] Ideally, gadget sampling can be performed on a lightweight device while other costly computations are done on a powerful machine and in an offline phase.

4.1 The Algorithm

Let us recall that $\mathbf{g} = (1, b, \cdots, b^{k-1})$, the modulus $q < b^k$ and as shown in [21] the gadget lattice $\Lambda^{\perp}(\mathbf{g}^t) = \{\mathbf{z} \in \mathbb{Z}^k : \mathbf{g}^t\mathbf{z} = 0 \mod q\}$ has a basis

$$\mathbf{B}_q = \begin{pmatrix} b & & & q_0 \\ -1 & b & & q_1 \\ & -1 & \ddots & \vdots \\ & & \ddots\ b & q_{k-2} \\ & & & -1\ q_{k-1} \end{pmatrix} = \begin{pmatrix} b & & & \\ -1 & b & & \\ & -1 & \ddots & \\ & & \ddots\ b & \\ & & & -1\ b \end{pmatrix} \begin{pmatrix} 1 & & & d_0 \\ & 1 & & d_1 \\ & & \ddots & \vdots \\ & & & 1\ d_{k-2} \\ & & & d_{k-1} \end{pmatrix} = \mathbf{TD}.$$

To sample $D_{\Lambda^{\perp}(\mathbf{g}^t),s,\mathbf{u}}$, our algorithm proceeds in two steps as per [39]. First, it generates a perturbation vector $\mathbf{p}$ of covariance $r^2\mathbf{\Sigma}_2 = s^2\mathbf{I}_k - r^2\mathbf{T}\mathbf{T}^t$. Then it samples $\mathbf{v}'$ from $D_{\mathcal{L}(\mathbf{D}),r,\mathbf{T}^{-1}(\mathbf{u}-\mathbf{p})}$ and the final output is $\mathbf{v} = \mathbf{T}\mathbf{v}'$. The second step is easily implemented over integers. To avoid floating point operations in perturbation sampling, we use a similar technique in [18]. Specifically, we discover a simple binary matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & & & 1 \\ & 1 & 1 & & \\ & & \ddots & \ddots & \\ & & & 1 & 1 \end{pmatrix} \in \mathbb{Z}^{k\times(k+2)}$$

such that $\mathbf{A}\begin{pmatrix} b\cdot\mathbf{I}_{k+1} & \\ & 1 \end{pmatrix}\mathbf{A}^t = (b+1)^2\cdot\mathbf{I}_k - \mathbf{T}\mathbf{T}^t = \mathbf{\Sigma}_2$ for $s = (b+1)r$ which coincides with [21]. According to Theorem 1, applying a linear transformation of $\mathbf{A}$ on some Gaussian of covariance $r^2\begin{pmatrix} b\cdot\mathbf{I}_{k+1} & \\ & 1 \end{pmatrix}$ gives the perturbation of covariance $r^2\mathbf{\Sigma}_2$. The matrix $\mathbf{A}$ has much less columns than the generic Gram decompositions in [18], which boosts the practical performance greatly. Additionally, the Gaussian transformed by $\mathbf{A}$ is non-spherical unlike the case in [18], which is crucial for non-square b. We formally describe our sampler in Algorithm 1 and prove its correctness in Lemma 4.

Algorithm 1: Gadget Gaussian sampler $\mathsf{GadgetGaussian}(b, k, q, l, s, u)$

Input: positive integers b, k, q, l such that $q < b^k$, $\mathbf{q} = [q]_b^k$ and $l \geq 4\sqrt{bk}$, $s = (b+1)r$ with $r \geq \overline{\eta_\epsilon}(\mathbb{Z}^k)$, $u \in \mathbb{Z}_q$ and $\mathbf{u} = [u]_b^k$.

Output: a sample $\mathbf{x}$ from a distribution within max-log distance $(2k+6)\hat{\epsilon}$ of $D_{\Lambda^\perp(\mathbf{g}^t), s, -\mathbf{u}}$

$d_0 = q_0/b$
for $i = 1, \cdots, k-1$ **do**
  $d_i = (d_{i-1} + q_i)/b$
end for
$\mathbf{p} \leftarrow \mathsf{Pert}(r, b, k, l)$ $\quad \{\mathbf{p} \sim D_{\mathbb{Z}^k/l, r\sqrt{(b+1)^2\mathbf{I}_k - \mathbf{T}\mathbf{T}^t}}\}$
$\mathbf{c} \leftarrow \mathbf{T}^{-1}(\mathbf{p} - \mathbf{u})$
$\mathbf{z} \leftarrow \mathsf{SampleD}(r, \mathbf{c}, \mathbf{d})$ $\quad \{\mathbf{Dz} \sim D_{\mathcal{L}(\mathbf{D}), r, \mathbf{c}}, \mathbf{D} = \begin{pmatrix} \mathbf{I}_{k-1} & \mathbf{d} \\ \mathbf{0} & \end{pmatrix}\}$
return $\mathbf{x} \leftarrow \mathbf{B}_q \mathbf{z}$

Algorithm 2: The subroutine $\mathsf{Pert}(r, b, k, l)$

Input: positive integers b, k, l such that $l \geq 4\sqrt{bk}$ and $r \geq \overline{\eta_\epsilon}(\mathbb{Z}^k)$.

Output: a sample $\mathbf{p}$ from a distribution within max-log distance $2\hat{\epsilon}$ of $D_{\mathbb{Z}^k/l, r\sqrt{(b+1)^2\mathbf{I}_k - \mathbf{T}\mathbf{T}^t}}$

$\mathbf{A} = \begin{pmatrix} 1 & 1 & & & 1 \\ & 1 & 1 & & \\ & & \ddots & \ddots & \\ & & & 1 & 1 \end{pmatrix} \in \mathbb{Z}^{k \times (k+2)}$
$\mathbf{y} \leftarrow (\bar{\mathbf{y}}, y_{k+1}) \in \mathbb{Z}^{k+2}$ where $\bar{\mathbf{y}} \leftarrow D_{\mathbb{Z}^{k+1}, lr\sqrt{b}}$, $y_{k+1} \leftarrow D_{\mathbb{Z}, lr}$
return $\mathbf{p} \leftarrow \frac{1}{l} \cdot \mathbf{A}\mathbf{y}$

Algorithm 3: The subroutine $\mathsf{SampleD}(r, \mathbf{c}, \mathbf{d})$

Input: vectors $\mathbf{c}, \mathbf{d}$ such that $\mathbf{D} = \begin{pmatrix} \mathbf{I}_{k-1} & \mathbf{d} \\ \mathbf{0} & \end{pmatrix}$, $r \geq \overline{\eta_\epsilon}(\mathbb{Z}^k)$.

Output: a sample $\mathbf{z}$ such that the distribution of $\mathbf{Dz}$ is within max-log distance $2k\hat{\epsilon}$ of $D_{\mathcal{L}(\mathbf{D}), r, \mathbf{c}}$

$z_{k-1} \leftarrow D_{\mathbb{Z}, r/d_{k-1}, c_{k-1}/d_{k-1}}$
$\mathbf{c} \leftarrow \mathbf{c} - z_{k-1}\mathbf{d}$
for $i = 0, \cdots, k-2$ **do**
  $z_i \leftarrow D_{\mathbb{Z}, r, c_i}$
end for
return $\mathbf{z}$

Lemma 4. *Let $b, k, q, l \in \mathbb{N}$ such that $q < b^k$ and $l \geq 4\sqrt{bk}$. Let $\epsilon \in (0, \frac{1}{2})$ and $s \geq (b+1) \cdot \overline{\eta_\epsilon}(\mathbb{Z}^k)$. For any $u \in \mathbb{Z}_q$ with $\mathbf{u} = [u]_b^k$, $\mathsf{GadgetGaussian}(b, k, q, l, s, u)$ returns a sample within a max-log distance $(2k+6)\hat{\epsilon}$ from $D_{\Lambda^\perp(\mathbf{g}^t), s, -\mathbf{u}}$.*

Remark 1. All involved base samplers are assumed to be perfect for simplicity. It is routine to adapt Lemma 4 to the setting of imperfect base samplers.

Proof. We first prove the correctness of $\mathsf{Pert}(r, b, k, l)$. Let $\mathbf{S} = \begin{pmatrix} \sqrt{b} \cdot \mathbf{I}_{k+1} & \\ & 1 \end{pmatrix}$ and $\mathbf{\Sigma}_2 = (b+1)^2\mathbf{I}_k - \mathbf{T}\mathbf{T}^t$. A routine computation shows $(\mathbf{AS})(\mathbf{AS})^t = \mathbf{\Sigma}_2$. Clearly, $\mathbf{y}$ follows $D_{\mathbb{Z}^{k+2}, rl\mathbf{S}}$ and $\mathbf{A}\mathbb{Z}^{k+2} = \mathbb{Z}^k$. Note that $\mathbb{Z}^{k+2} \cap \mathsf{ker}(\mathbf{A})$ is a lattice of rank 2 and contains two linearly independent vectors $\mathbf{v}_0 = (1, 0, \cdots, 0, -1), \mathbf{v}_1 = ((-1)^0, (-1)^1, \cdots, (-1)^k, 0)$ of norm $\leq \sqrt{k+1}$. According to Lemma 1, it holds that $\eta_\epsilon(\mathbb{Z}^{k+2} \cap \mathsf{ker}(\mathbf{A})) \leq \overline{\eta_\epsilon}(\mathbb{Z}^2)\sqrt{k+1} \leq rl$ and then $\eta_\epsilon(\mathbb{Z}^{k+2} \cap \mathsf{ker}(\mathbf{A})) \leq rl\mathbf{S}$ as $\mathbf{S} \geq \mathbf{I}$. By Theorem 1, we have

$$\Delta_{ML}(\mathbf{p}, D_{\mathbb{Z}^k/l, r\sqrt{\mathbf{\Sigma}_2}}) = \Delta_{ML}(l\mathbf{p}, D_{\mathbb{Z}^k, rl\mathbf{AS}}) = \Delta_{ML}(\mathbf{Ay}, D_{\mathbb{Z}^k, rl\mathbf{AS}}) \leq 2\hat{\epsilon}$$

and the correctness of $\mathsf{Pert}(r, b, k, l)$ follows.

Since $\|\tilde{\mathbf{D}}\|_{col} \leq 1$, Lemma 1 shows $s \geq (b+1)\eta_\epsilon(\mathbf{D})$. The algorithm $\mathsf{SampleD}(r, \mathbf{c}, \mathbf{d})$ is actually Klein algorithm [25] on $\mathbf{D}$, so $\Delta_{ML}(\mathbf{Dz}, D_{\mathcal{L}(\mathbf{D}), r, \mathbf{c}}) \leq 2k\hat{\epsilon}$ by Theorem 4.1 of [25] and $\|\tilde{\mathbf{D}}\|_{col} \leq 1$. It remains to show $r\sqrt{\mathbf{\Sigma}_3} \geq \eta_\epsilon(\mathbb{Z}^k/l)$ where $\mathbf{\Sigma}_3^{-1} = \mathbf{\Sigma}_1^{-1} + \mathbf{\Sigma}_2^{-1}$ and $\mathbf{\Sigma}_1 = \mathbf{T}\mathbf{T}^t$ as per Theorem 2. Indeed [21] showed in Corollary 1 that $r'\sqrt{\mathbf{\Sigma}_3} \geq \eta_\epsilon(\mathcal{L}(\mathbf{\Sigma}_2))$ for $r' = \frac{\sqrt{2b}(2b+1)}{b+1}\overline{\eta_\epsilon}(\mathbb{Z}^k)$. From $\mathcal{L}(\mathbf{\Sigma}_2) \subset \mathbb{Z}^k$, it follows that $\eta_\epsilon(\mathbb{Z}^k) \leq \eta_\epsilon(\mathcal{L}(\mathbf{\Sigma}_2))$ and then

$$\eta_\epsilon(\mathbb{Z}^k/l) \leq \frac{\eta_\epsilon(\mathcal{L}(\mathbf{\Sigma}_2))}{l} \leq \frac{r'}{l}\sqrt{\mathbf{\Sigma}_3} \leq r\sqrt{\mathbf{\Sigma}_3}.$$

We now complete the proof. □

4.2 Comparison

The comparison between our gadget Gaussian sampler and the Genise-Micciancio one [21] is summarized as follows:

Gaussian quality. Both two samplers are proposed to sample a spherical Gaussian over the gadget lattice. The quality of the sampler is measured by the minimal Gaussian width s it achieves. As shown in Lemma 4, the minimal s for our sampler is $(b+1) \cdot \overline{\eta_\epsilon}(\mathbb{Z}^k)$. While a lower bound of s given in [21] (Corollary 1) is $\sqrt{2b}(2b+1) \cdot \overline{\eta_\epsilon}(\mathbb{Z}^k)$, it is improved to $(b+1) \cdot \overline{\eta_\epsilon}(\mathbb{Z}^k)$ in [16] via replacing the integer perturbation sampling with a continuous version. Therefore, our sampler achieves the same quality with the Genise-Micciancio one.

Arithmetic. All intermediate numbers in Algorithm 1 are either integer or fraction with a simple bounded denominator, which supports a complete integer implementation. Indeed some base samplers (for $D_{\mathbb{Z}, lr\sqrt{b}}$ and $D_{\mathbb{Z}, r/d_{k-1}, c_{k-1}/d_{k-1}}$) deal with irrational width or relatively complicated center. Nevertheless, they can still be implemented over integers by classic techniques [38, 6]. In contrast, the Genise-Micciancio sampler needs floating point arithmetic in computing a square Gram root of $\mathbf{\Sigma}_2$, and to achieve higher quality, it also requires continuous Gaussian samplings.

Memory. As a direct result of integer arithmetic, our sampler requires less RAM and storage for precomputed values. In addition, the new-introduced matrix $\mathbf{A}$ is of regular structure and thus causes no storage overhead.

Time complexity. Algorithm 1 consists of $(2k + 2)$ integer samplings and other arithmetic computations, i.e. computing $\mathbf{d}, \mathbf{Ay}, \mathbf{c}$, need only $O(k)$ integer operations thanks to the nice structures of $\mathbf{A}, \mathbf{T}$. Therefore our sampler runs in $O(k)$ assuming constant time for base samplings and scalar arithmetic, which is the same with the Genise-Micciancio sampler. Additionally, the subroutine Algorithm 2 is highly parallelizable.

Experimental result. We implement our new sampler and compare with the implementations of the Genise-Micciancio sampler and its variant in [16] available in the PALISADE library[2]. For a fair comparison, we implement all base samplers with the open source code of Karney sampler. The experiments were run in C++ on a laptop with an Intel Core i7-10510U CPU with 4 cores @ 1.80GHz, running Ubuntu 20.04.2 LTS.

Figure 1 shows the speed comparison among three algorithms under different moduli q and the same base $b = 2$ and width $s = 100$. Basically, our algorithm is as fast as the Genise-Micciancio sampler but about twice slower than the variant in [16]. Figure 2 shows the speed comparison under different bases b and a fixed modulus $q \approx 9 \cdot 10^{18}$. In the corresponding experiment, we work with $s = (b + 1) \cdot \overline{\eta_\epsilon}(\mathbb{Z}^k) \approx 4.578(b + 1)$ as used in practice [16]. Since the bound of s in [21] is greatly larger than that in the variant of [16], we omit the comparison with [21]. The samplers in [21] and [16] need floating-point arithmetic for Cholesky decomposition, and the one in [16] also uses continuous Gaussian sampling. In contrast, our algorithm avoids all floating-point operations. The efficiency advantage of the variant of [16] is due to the fact that the continuous Gaussian sampling in C++ header file "random" is significantly faster than the Karney sampling in the PALISADE library. With a faster base sampler, our algorithm hopefully outperforms the one of [16].

[2] https://palisade-crypto.org/

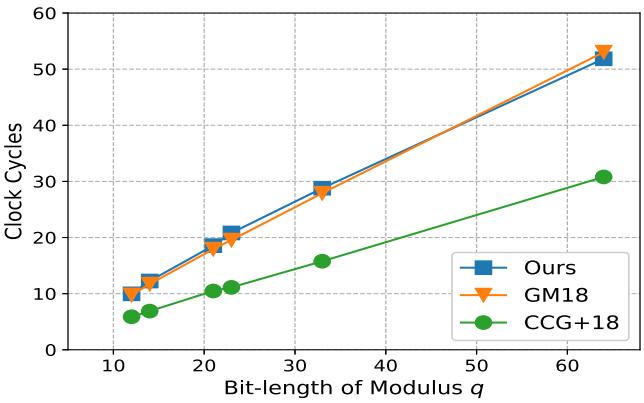

Fig. 1. Measured clock cycles with $q \in \{4093, 12289, 1676083, 8383498, 4295967357, \approx 9 \cdot 10^{18}\}$, $b = 2$ and $s = 100$ averaged over 1000,000 runs.

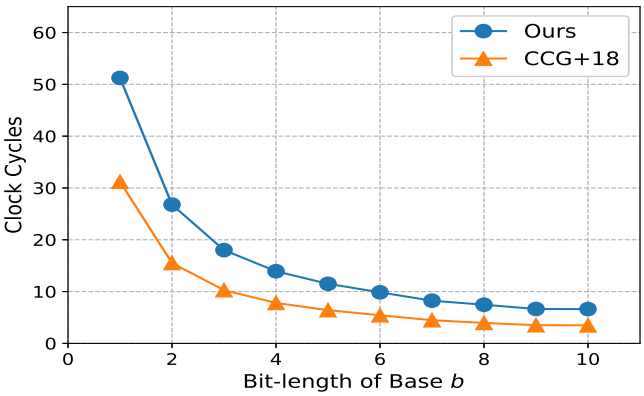

Fig. 2. Measured clock cycles with $b \in \{2^1, 2^2, \cdots, 2^{10}\}$, $q \approx 9 \cdot 10^{18}$ and $s = (b+1) \cdot \overline{\eta_\epsilon}(\mathbb{Z}^k)$ averaged over 1000,000 runs.

5 Improved Gadget Subgaussian Sampling

In this section, we present a new gadget subgaussian sampler for arbitrary moduli. Compared with the Genise-Micciancio-Polyakov algorithm [23] (See Section 3.2), our algorithm is simpler, faster and requires asymptotically less randomness. As for the quality, while the subgaussian parameter achieved by our

sampler is $\sqrt{2}$ times as large as that by the Genise-Micciancio-Polyakov sampler for large b and in the worst case, the actual quality of our sampler is close to even better than that of the Genise-Micciancio-Polyakov sampler for practical parameters (q, b).

5.1 The Algorithm

Our algorithm distinguishes two cases of $q = b^k$ and $q < b^k$. For $q = b^k$, the sampling is identical to the existing algorithm (Algorithm 1, [23]) that is easy and efficient. But for $q < b^k$, our sampler proceeds very differently: it fully exploits the ease of the procedure for $q = b^k$ and does not use special linear transformation as the existing approach (Algorithm 2, [23]). The idea stems from a simple observation that for $q = b^k$ and $u \in \mathbb{Z}_q$, the output $\mathbf{x}$ satisfies $\langle \mathbf{x}, \mathbf{g} \rangle \in \{u, u - q\}$ (See Lemma 6). The values u and $u - q$ basically determine the most significant digit x_{k-1} and thus $\langle \mathbf{x}', \mathbf{g}' \rangle \bmod b^{k-1}$ where $\mathbf{x}' = (x_0, x_1 \cdots, x_{k-2})$ and $\mathbf{g}' = (1, b, \cdots, b^{k-2})$. Our sampler for $q < b^k$ is designed upon above facts; it consists of three steps: first, to choose $\langle \mathbf{x}', \mathbf{g}' \rangle \bmod b^{k-1}$ according to proper probability; then, to sample a subgaussian $\mathbf{x}'$ with the sampler for $q = b^{k-1}$ given $\langle \mathbf{x}', \mathbf{g}' \rangle \bmod b^{k-1}$; finally, to determine the last coefficient x_{k-1} as per $\mathbf{x}'$. The formal description is illustrated in Algorithm 4.

Lemma 5 shows the correctness and performance of Algorithm 4.

Lemma 5. *Let $b, k, q, u \in \mathbb{N}$ such that $q \leq b^k$ and $u \in \mathbb{Z}_q$. Then $\mathsf{SubGaussian}(b, k, q, u)$ outputs a subgaussian vector over $\Lambda_u^{\perp}(\mathbf{g}^t)$. More precisely,*

- *if $q = b^k$, $\mathsf{SubGaussian}(b, k, q, u)$ uses $\log q$ random bits, runs in $O(k)$ time and space and achieves subgaussian parameter at most $(b-1)\sqrt{2\pi}$;*
- *if $q < b^k$, $\mathsf{SubGaussian}(b, k, q, u)$ uses $\log q + (k-1)\log b$ random bits, runs in $O(k)$ time and space and achieves subgaussian parameter at most $\sqrt{(b-1)^2 + \alpha^2}\sqrt{2\pi}$ with $\alpha = \lfloor q/b^{k-1} \rfloor + 1$.*

Remark 2. As a by-product, a digit decomposition for an arbitrary modulus is obtained by de-randomizing Algorithm 4, that is replacing lines 8 and 17 with deterministically choosing the option of higher probability. It can be seen that the output of this digit decomposition is of infinity norm $\leq b/2$.

To prove Lemma 5, we need the following lemma.

Lemma 6. *Let $b, k, q, u \in \mathbb{N}$ such that $q = b^k$ and $u \in \mathbb{Z}_q$. Let $\mathbf{x}$ be the output of $\mathsf{SubGaussian}(b, k, q = b^k, u)$ and $\mathbf{g} = (1, b, \cdots, b^{k-1})$. Then $\langle \mathbf{x}, \mathbf{g} \rangle = u$ with probability $(q - u)/q$; $\langle \mathbf{x}, \mathbf{g} \rangle = u - q$ with probability u/q.*

Proof. Since $|x_i| < b$, some simple computation yields that $\langle \mathbf{x}, \mathbf{g} \rangle \in (-b^k, b^k)$. Together with the fact that $\langle \mathbf{x}, \mathbf{g} \rangle = u \bmod q$, it follows that $\langle \mathbf{x}, \mathbf{g} \rangle \in \{u, u - q\}$. Let p denote the probability of $\langle \mathbf{x}, \mathbf{g} \rangle = u$, then

$$\mathbb{E}[\langle \mathbf{x}, \mathbf{g} \rangle] = p \cdot u + (1 - p)(u - q).$$

Algorithm 4: Gaussian subgaussian sampler $\mathsf{SubGaussian}(b, k, q, u)$

Input: positive integers b, k, q, u such that $q \leq b^k$ and $u \in \mathbb{Z}_q$.
Output: subgaussian $\mathbf{x} \in \Lambda_u^\perp(\mathbf{g}^t)$ with parameter $(b-1)\sqrt{2\pi}$ when $q = b^k$; with parameter $\sqrt{(b-1)^2 + \alpha^2}\sqrt{2\pi}$ with $\alpha = \lfloor q/b^{k-1} \rfloor + 1$ when $q < b^k$.

if $q = b^k$ **then**
  $\mathbf{x} \leftarrow \mathbf{0}$
  for $i = 0, \cdots, k-1$ **do**
    $y \leftarrow u \mod b \in \{0, \cdots, b-1\}$
    if $y = 0$ **then**
      $x_i \leftarrow 0$
    else
      with probability y/b, $x_i \leftarrow y - b$, and $x_i \leftarrow y$ otherwise
    end if
    $u \leftarrow (u - x_i)/b$
  end for
  return $\mathbf{x}$
end if
$u_0 \leftarrow u \bmod b^{k-1}$, $u_1 \leftarrow (u - q) \bmod b^{k-1}$
$a_0 \leftarrow \lfloor \frac{u}{b^{k-1}} \rfloor$, $a_1 \leftarrow \lfloor \frac{u-q}{b^{k-1}} \rfloor$
sample r uniformly over $[0, 1]$
if $r < \frac{q-u}{q}$ **then**
  $\mathbf{x}' \leftarrow \mathsf{SubGaussian}(b, k-1, b^{k-1}, u_0)$
  if $\langle \mathbf{x}', \mathbf{g}' \rangle = u_0$ with $\mathbf{g}' = (1, b, \cdots, b^{k-2})$ **then**
    return $\mathbf{x} = (\mathbf{x}', a_0)$ $\quad \{\langle \mathbf{x}', \mathbf{g}' \rangle = u_0\}$
  else
    return $\mathbf{x} = (\mathbf{x}', a_0 + 1)$ $\quad \{\langle \mathbf{x}', \mathbf{g}' \rangle = u_0 - b^{k-1}\}$
  end if
else
  $\mathbf{x}' \leftarrow \mathsf{SubGaussian}(b, k-1, b^{k-1}, u_1)$
  if $\langle \mathbf{x}', \mathbf{g}' \rangle = u_1$ **then**
    return $\mathbf{x} = (\mathbf{x}', a_1)$ $\quad \{\langle \mathbf{x}', \mathbf{g}' \rangle = u_1\}$
  else
    return $\mathbf{x} = (\mathbf{x}', a_1 + 1)$ $\quad \{\langle \mathbf{x}', \mathbf{g}' \rangle = u_1 - b^{k-1}\}$
  end if
end if

At each step, x_i is chosen from $\{y, y-b\}$ with expectation 0. Therefore

$$p \cdot u + (1-p)(u-q) = \mathbb{E}[\langle \mathbf{x}, \mathbf{g} \rangle] = \sum_{i=0}^{k-1} b^i \cdot \mathbb{E}[x_i] = 0.$$

This shows $p = (q-u)/q$ and the proof is completed. □

Proof of Lemma 5. For the case $q = b^k$, Algorithm 4 is the same with Algorithm 1 in [23]. By Theorem 4 of [23], the statement for $q = b^k$ is proved. It remains to prove the statement for $q < b^k$.

To this end, we first prove that the output $\mathbf{x}$ satisfies $\langle \mathbf{x}, \mathbf{g} \rangle = u \bmod q$. Lemma 6 shows that $\langle \mathbf{x}', \mathbf{g}' \rangle \in \{u_{bit}, u_{bit} - b^{k-1}\}$ for $bit \in \{0, 1\}$. When $\langle \mathbf{x}', \mathbf{g}' \rangle = u_{bit}$, it holds that $x_{k-1} = a_{bit}$ and thus

$$\langle \mathbf{x}, \mathbf{g} \rangle = \langle \mathbf{x}', \mathbf{g}' \rangle + x_{k-1} \cdot b^{k-1} = u_{bit} + a_{bit} \cdot b^{k-1} = u - bit \cdot q.$$

When $\langle \mathbf{x}', \mathbf{g}' \rangle = u_{bit} - b^{k-1}$, it holds that $x_{k-1} = a_{bit} + 1$ and thus

$$\langle \mathbf{x}, \mathbf{g} \rangle = \langle \mathbf{x}', \mathbf{g}' \rangle + x_{k-1} \cdot b^{k-1} = u_{bit} - b^{k-1} + (a_{bit} + 1) \cdot b^{k-1} = u - bit \cdot q.$$

Therefore $\langle \mathbf{x}, \mathbf{g} \rangle = u \bmod q$ always holds.

Next, we show that $\mathbb{E}[x_{k-1}] = 0$ and $|x_{k-1}| \leq \alpha$, so that the random variable x_{k-1} is subgaussian with parameter $\alpha\sqrt{2\pi}$. Indeed as shown in Algorithm 4, x_{k-1} only has four possible values $\{a_0, a_0+1, a_1, a_1+1\}$. Since $u \in \mathbb{Z}_q$, we have that $a_0 = \lfloor \frac{u}{b^{k-1}} \rfloor \in \left[0, \lfloor \frac{q}{b^{k-1}} \rfloor\right] = [0, \alpha - 1]$ and $a_1 = \lfloor \frac{u-q}{b^{k-1}} \rfloor \in \left[\lfloor \frac{-q}{b^{k-1}} \rfloor, -1\right] = [-\alpha, -1]$. Immediately, $|x_{k-1}| \leq \alpha$. As for $\mathbb{E}[x_{k-1}]$, we note that $x_{k-1} = a_0$ occurs if and only if $r < \frac{q-u}{q}$ and $\langle \mathbf{x}', \mathbf{g}' \rangle = u_0$. By Lemma 6, it follows that

$$\Pr[x_{k-1} = a_0] = \frac{(q-u)(b^{k-1} - u_0)}{q \cdot b^{k-1}}.$$

Similarly,

$$\Pr[x_{k-1} = a_0{+}1] = \frac{(q-u) \cdot u_0}{q \cdot b^{k-1}}; \ \Pr[x_{k-1} = a_1] = \frac{u \cdot (b^{k-1} - u_1)}{q \cdot b^{k-1}}; \ \Pr[x_{k-1} = a_1{+}1] = \frac{u \cdot u_1}{q \cdot b^{k-1}}.$$

Thus we have

$$\begin{aligned}
\mathbb{E}[x_{k-1}] &= a_0 \frac{(q-u)(b^{k-1} - u_0)}{q \cdot b^{k-1}} + (a_0 + 1) \frac{(q-u) \cdot u_0}{q \cdot b^{k-1}} + a_1 \frac{u \cdot (b^{k-1} - u_1)}{q \cdot b^{k-1}} + (a_1 + 1) \frac{u \cdot u_1}{q \cdot b^{k-1}} \\
&= \frac{(q-u)(a_0 \cdot b^{k-1} + u_0)}{q \cdot b^{k-1}} + \frac{u \cdot (a_1 \cdot b^{k-1} + u_1)}{q \cdot b^{k-1}} \\
&= \frac{(q-u)u}{q \cdot b^{k-1}} + \frac{u \cdot (u-q)}{q \cdot b^{k-1}} = 0
\end{aligned}$$

Then we verify $\mathbf{x}$ is subgaussian with parameter $\sqrt{(b-1)^2 + \alpha^2}\sqrt{2\pi}$. That is to show that $\langle \mathbf{x}, \mathbf{v} \rangle$ is subgaussian with parameter $\sqrt{(b-1)^2 + \alpha^2}\sqrt{2\pi}$ for all unit vectors $\mathbf{v} = (v_0, \cdots, v_{k-1})$. Let $\mathbf{v}' = (v_0, \cdots, v_{k-2})$. If $|v_{k-1}| = 1$, then

$\langle \mathbf{x}, \mathbf{v} \rangle = x_{k-1} v_{k-1}$ is subgaussian with parameter $\alpha\sqrt{2\pi}$ as per above argument. If $v_{k-1} = 0$, then $\langle \mathbf{x}, \mathbf{v} \rangle = \langle \mathbf{x}', \mathbf{v}' \rangle$ and $\mathbf{v}'$ is a unit vector. As per Algorithm 4 and the statement for $q = b^k$, $\mathbf{x}'$ is subgaussian with parameter $(b-1)\sqrt{2\pi}$ and thus $\langle \mathbf{x}, \mathbf{v} \rangle$ is subgaussian with parameter $(b-1)\sqrt{2\pi}$ if $v_{k-1} = 0$. For the case $0 < |v_{k-1}| < 1$, let $p_0 = \frac{1}{1-v_{k-1}^2}, p_1 = \frac{1}{v_{k-1}^2}$, then $\frac{1}{p_0} + \frac{1}{p_1} = 1$. By Hölder inequality, we have

$$\begin{aligned}
\mathbb{E}[e^{2\pi t \langle \mathbf{x}, \mathbf{v} \rangle}] =& \mathbb{E}[e^{2\pi t \langle \mathbf{x}', \mathbf{v}' \rangle + 2\pi t x_{k-1} v_{k-1}}] \\
\leq& \left[\mathbb{E}[e^{2\pi t \langle \mathbf{x}', \mathbf{v}' \rangle}]^{p_0}\right]^{1/p_0} \left[\mathbb{E}[e^{2\pi t x_{k-1} v_{k-1}}]^{p_1}\right]^{1/p_1} \\
\leq& \exp\left(2\pi^2 t^2 [(b-1)^2 (1 - v_{k-1}^2) p_0 + \alpha^2 v_{k-1}^2 p_1)]\right) \\
=& \exp\left(2\pi^2 t^2 ((b-1)^2 + \alpha^2)\right).
\end{aligned}$$

In summary, we prove that $\mathbf{x}$ is subgaussian with parameter $\sqrt{(b-1)^2 + \alpha^2}\sqrt{2\pi}$.

It is clear that the complexity of Algorithm 4 is $O(k)$. The random bits are used in two places: line 17 uses $\log q$ bits to determine $\langle \mathbf{x}', \mathbf{g}' \rangle$ mod b^{k-1} and the subroutine $\mathsf{SubGaussian}(b, k-1, b^{k-1}, u_{bit})$ uses $(k-1)\log b$ bits to output $\mathbf{x}'$; thus $\log q + (k-1)\log b$ random bits are used in total. The proof is completed. □

5.2 Comparison

In this subsection, we compare our new gadget subgaussian algorithm with the Genise-Micciancio-Polyakov one [23]. The comparison is restricted to the case $q < b^k$.

Randomness. Less randomness is one of the main advantages of subgaussian sampling. The Genise-Micciancio-Polyakov algorithm uses $k \log q = O(k^2 \log b)$ random bits, which was claimed to be "almost optimal" in [23]. In fact, our algorithm only needs $\log q + (k-1)\log b = O(k \log b)$ random bits. The reduced randomness is due to the fully use of the randomness-efficient subroutine for $q = b^{k-1}$ in which each coefficient consumes $\log b$ bits; in contrast, the i-th coefficient (before linear transformation) consumes $i \log b$ bits in Genise-Micciancio-Polyakov sampler. Notably, Algorithm 4 for $q < b^k$ needs an asymptotically same amount of randomness with the one for $q = b^k$. We therefore believe that it is essentially optimal in randomness requirement.

Complexity and performance. Both the Genise-Micciancio-Polyakov algorithm and ours achieve $O(k)$ complexity in time and space. Nevertheless, our sampler proceeds in a direct and simple way, which actually saves the computation and storage with respect to the complicated linear transformation.

We implement Algorithm 4 fully over integers in C++. Since the implementation of the Genise-Micciancio-Polyakov sampler in the PALISADE library uses floating-point arithmetic, we also adapt it to a fully integer version for better comparison. The gadget base b is restricted to a power-of-2 in the experiment, which leads to faster and more convenient operations as verified in [23].

The experiment environment was a laptop with an Intel Core i7-10510U CPU with 4 cores @ 1.80GHz, running Ubuntu 20.04.2 LTS. Figure 3 exhibits the practical performance of subgaussian samplers. It can be seen that our subgaussian sampler is faster than the Genise-Micciancio-Polyakov one whose the integer implementation outperforms the floating-point implementation in the PALISADE library. The speed of both algorithms mainly depends on the dimension $k = \lceil 60/\log b \rceil$. When $b = 2$ and $k = 60$, our algorithm is around 3.2 (resp. 2.3) times as fast as the PALISADE (resp. integer) implementation of the Genise-Micciancio-Polyakov algorithm. As k decreases, the speed advantage declines.

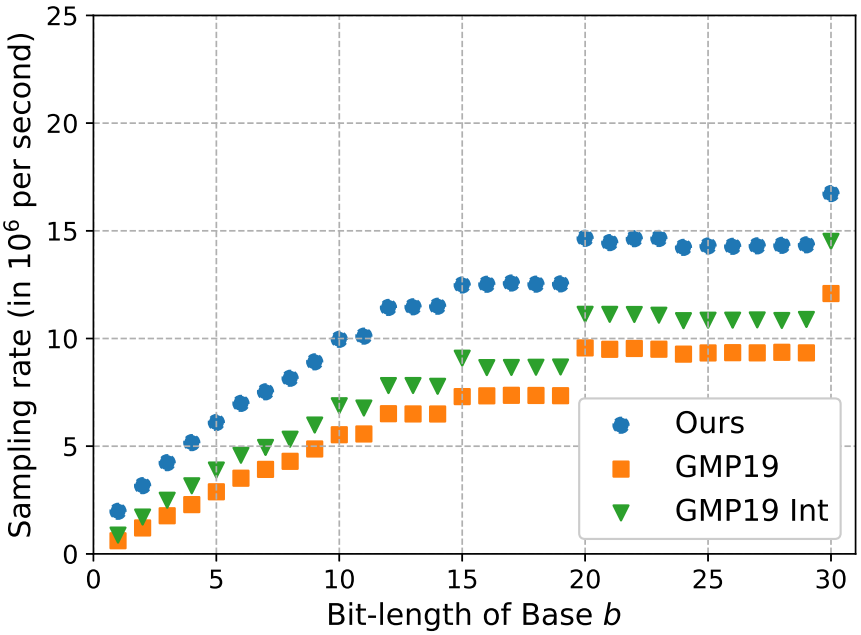

Fig. 3. Runtime of subgaussian sampling rate for native uniformly random integers (w.r.t a 60-bit modulus). Experimental values measure over 10^8 samplings.

Quality. The quality of the subgaussian sampler is measured by the subgaussian parameter it achieves. That is $Q_{our} = \sqrt{(b-1)^2 + \alpha^2}\sqrt{2\pi}$ for our sampler where $\alpha = \lfloor q/b^{k-1} \rfloor + 1 \leq b$ and $Q_{GMP} = (b+1)\sqrt{2\pi}$ for the Genise-Micciancio-Polyakov one. While $Q_{our} \approx \sqrt{2} \cdot Q_{GMP}$ in the worst case ($\alpha = b$) for large b, our sampler can get close and even better quality in some typical situations:

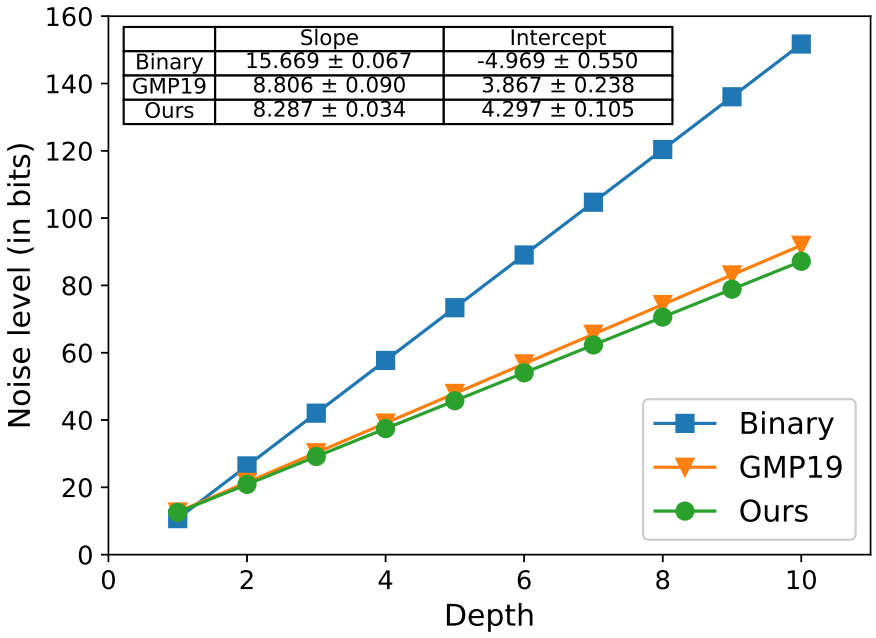

Fig. 4. Noise growth for GSW-type multiplication in the KP-ABE variant over $\mathbb{Z}[x]/(x^n+1)$ $(n = 1024, b = 2, k = 180)$. The slope of the linear interpolation is $\beta \log(mn)$ and β describes the rate of noise growth.

- for $b = 2$, the worst-case $Q_{our} = \sqrt{5}\sqrt{2\pi}$ is less than $Q_{GMP} = 3\sqrt{2\pi}$. For a visualized comparison, we examine the effect of our algorithm (with $b = 2$) on the noise growth in GSW-type products [26], which is a typical application of subgaussian sampling. In the experiment, we generate a random error vector in $\mathcal{R}_q^m$ where $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$ and $m = k+2$ and then iteratively multiply it by a matrix $(\mathbf{g}^{-1}(u_i)) \in \mathcal{R}^{m\times m}$ in which $\mathbf{g}^{-1}(u_i) \in \mathcal{R}^m$ denotes the output of either subgaussian or binary decomposition with input $u_i \in \mathcal{R}_q$. The noise level (in bits) grows almost linearly in the depth, and the noise growth rate is $(mn)^\beta$. As shown in Figure 4, our algorithm achieves $\beta \approx 8.287/\log(mn) \approx 0.47$ less than 0.50 and 0.89 for the Genise-Micciancio-Polyakov one and the common binary decomposition, which means our subgaussian algorithm may lead to more compact parameters in some advanced applications.
- for a large base b, there exist a certain number of NTT moduli q such that $Q_{our} \leq 1.05 \cdot Q_{GMP}$. Moreover, some of these moduli can even achieve such a bounded Q_{our} for all possible b's. Table 1 shows five such NTT moduli and corresponding Q_{our}/Q_{GMP} with different b.

Overall, by choosing proper (q, b), it is convenient and flexible to make our sampler achieve a similar quality with the Genise-Micciancio-Polyakov one in practical use cases without efficiency and security loss.

Remark 3. The quality of the Genise-Micciancio-Polyakov sampler is determined by the maximal singular value of the used linear transformation $\mathbf{T}$ (Eq. (2)), and independent of the modulus q. As k grows, the maximal singular value of $\mathbf{T}$ converges to $(b+1)$ as shown in [33]. Therefore, we fix $Q_{GMP} = (b+1)\sqrt{2\pi}$ as a tight bound for the quality of the Genise-Micciancio-Polyakov sampler.

Remark 4. Despite the different subgaussian parameters, for both the Genise-Micciancio-Polyakov sampler and ours, the infinity norm of the output vector is bounded by b.

Table 1. The values of $\frac{Q_{our}}{Q_{GMP}}$ for some recommended NTT moduli.

b \ q	$2^{22}+2^{13}+2^{12}+1$	$2^{30}+2^{13}+1$	$2^{40}+2^{15}+2^{14}+2^{11}+1$	$2^{52}+2^{16}+2^{13}+2^{11}+1$	$2^{60}+2^{15}-2^{11}+1$
2	0.745	0.745	0.745	0.745	0.745
2^2	0.721	0.721	0.721	0.721	0.721
2^3	0.846	0.808	0.846	0.846	0.808
2^4	0.930	0.930	0.890	0.890	0.890
2^5	0.951	0.941	0.941	0.951	0.941
2^6	1.004	0.969	1.003	1.003	0.969
2^7	0.984	0.985	1.017	0.986	0.993
2^8	1.023	1.023	0.992	0.994	0.994
2^9	0.996	0.996	0.996	1.027	1.004
2^{10}	0.998	0.998	0.998	0.998	0.998
2^{11}	0.999	1.006	1.001	1.006	0.999
2^{12}	1.030	0.999	1.030	0.999	0.999
2^{13}		0.999	0.999	0.999	1.000
2^{14}		0.999	1.030	1.001	0.999
2^{15}		0.999	1.000	0.999	0.999
2^{16}		1.030	0.999	0.999	1.001
2^{17}			0.999	0.999	0.999
2^{18}			0.999	1.030	0.999
2^{19}			0.999	1.000	0.999
2^{20}			0.999	1.000	0.999
2^{21}			1.030	0.999	1.007
2^{22}				0.999	1.000
2^{23}				0.999	1.000
2^{24}				0.999	0.999
2^{25}				0.999	0.999
2^{26}				0.999	0.999
2^{27}				1.030	0.999
2^{28}					0.999
2^{29}					0.999
2^{30}					0.999
2^{31}					1.030

6 Conclusion

To conclude, we develop new gadget Gaussian and subgaussian sampling algorithms. Our gadget Gaussian sampler for arbitrary moduli gets rid of the

reliance on high-precision arithmetic while keeping a good efficiency and quality. It can be a potentially more efficient option for gadget sampling in the context of constrained environments and side-channel countermeasures. Additionally, our gadget subgaussian sampler is simpler, faster and needs asymptotically less randomness compared with the previous result. For practical parameters, it also achieves almost the same quality with the previous sampler. Hence it should be a refined alternative to the current subgaussian algorithm. Overall our results provide the current lattice gadget toolkit with some simpler and efficient algorithm candidates, and improve the practicality of the gadget toolkit.

6.1 Future work

In this work, we focus on the gadget algorithms associated to the typical gadget $\mathbf{g} = (1, b, \cdots, b^{k-1})$. Some lattice applications [31, 10, 23] use the CRT gadget to improve the efficiency. The CRT gadget is a generalized gadget based on the Chinese Remainder Theorem, which is particularly effective for very large moduli. The algorithms for $\mathbf{g} = (1, b, \cdots, b^{k-1})$ can be directly adapted to the CRT form, thus we omit the related details. Nevertheless, it would be worthy to implement and evaluate our algorithms in the CRT setting.

The main interest of this work is the fundamental algorithms themselves, and we do not study deeply from an implementation aspect. With the post-quantum standardization underway, implementing more powerful lattice cryptosystems may gain increasingly attention. We leave the optimized implementation and the application of our results to practical implementations of lattice schemes as future works. Additionally, our subgaussian sampler and the Genise-Micciancio-Polyakov one seem susceptible to timing leakage. While this leakage is not an issue in most current applications, the side-channel protections of gadget algorithms require a future investigation.

While a general definition of gadget was proposed in [23], almost all known gadget algorithms are designed for the gadget $\mathbf{g} = (1, b, \cdots, b^{k-1})$ and its CRT generalization. To develop more practical gadgets and associated algorithms is an interesting problem.

Acknowledgements. We thank Léo Ducas for his helpful comments. This work is supported by the National Key Research and Development Program of China (Grant No. 2018YFA0704701), the Major Program of Guangdong Basic and Applied Research(Grant No. 2019B030302008) and Major Scientific and Techological Innovation Project of Shandong Province, China (Grant No. 2019JZZY010133).

References

[1] Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: EUROCRYPT 2010. pp. 553–572 (2010)

[2] Agrawal, S., Boyen, X., Vaikuntanathan, V., Voulgaris, P., Wee, H.: Functional encryption for threshold functions (or fuzzy IBE) from lattices. In: PKC 2012. pp. 280–297 (2012)
[3] Agrawal, S., Freeman, D.M., Vaikuntanathan, V.: Functional encryption for inner product predicates from learning with errors. In: ASIACRYPT 2011. pp. 21–40. Springer (2011)
[4] Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: CRYPTO 2014. pp. 297–314 (2014)
[5] Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986). https://doi.org/10.1007/BF02579403, https://doi.org/10.1007/BF02579403
[6] Barthe, G., Belaïd, S., Espitau, T., Fouque, P.A., Rossi, M., Tibouchi, M.: GALACTICS: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited. In: ACM CCS 2019. pp. 2147–2164 (2019)
[7] Bellare, M., Kiltz, E., Peikert, C., Waters, B.: Identity-based (lossy) trapdoor functions and applications. In: EUROCRYPT 2012. pp. 228–245 (2012)
[8] Bert, P., Eberhart, G., Prabel, L., Roux-Langlois, A., Sabt, M.: Implementation of lattice trapdoors on modules and applications. In: PQCRYPTO 2021. pp. 195–214 (2021)
[9] Bert, P., Fouque, P.A., Roux-Langlois, A., Sabt, M.: Practical implementation of Ring-SIS/LWE based signature and IBE. In: PQCrypto 2018. pp. 271–291 (2018)
[10] Bonnoron, G., Ducas, L., Fillinger, M.: Large FHE gates from tensored homomorphic accumulator. In: AFRICACRYPT 2018. pp. 217–251. Springer (2018)
[11] Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: STOC 2013. pp. 575–584 (2013)
[12] Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM Journal on Computing **43**(2), 831–871 (2014)
[13] Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: EUROCRYPT 2010. pp. 523–552 (2010)
[14] Chen, Y., Genise, N., Mukherjee, P.: Approximate trapdoors for lattices and smaller hash-and-sign signatures. In: ASIACRYPT 2019 (2019), to appear
[15] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
[16] Cousins, D.B., Crescenzo, G.D., Gür, K.D., King, K., Polyakov, Y., Rohloff, K., Ryan, G.W., Savas, E.: Implementing conjunction obfuscation under entropic ring lwe. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 354–371 (2018)
[17] Dai, W., Doröz, Y., Polyakov, Y., Rohloff, K., Sajjadpour, H., Savaş, E., Sunar, B.: Implementation and evaluation of a lattice-based key-policy ABE scheme. IEEE Transactions on Information Forensics and Security **13**(5), 1169–1184 (2018)

[18] Ducas, L., Galbraith, S., Prest, T., Yu, Y.: Integral Matrix Gram Root and Lattice Gaussian Sampling without Floats. In: EUROCRYPT 2020. pp. 608–637 (2020)
[19] Ducas, L., Lyubashevsky, V., Prest, T.: Efficient identity-based encryption over NTRU lattices. In: ASIACRYPT 2014. pp. 22–41 (2014)
[20] Ducas, L., Prest, T.: Fast fourier orthogonalization. In: ISSAC 2016. pp. 191–198 (2016)
[21] Genise, N., Micciancio, D.: Faster gaussian sampling for trapdoor lattices with arbitrary modulus. In: EUROCRYPT 2018. pp. 174–203 (2018)
[22] Genise, N., Micciancio, D., Peikert, C., Walter, M.: Improved discrete gaussian and subgaussian analysis for lattice cryptography. In: PKC 2020. pp. 623–651 (2020)
[23] Genise, N., Micciancio, D., Polyakov, Y.: Building an efficient lattice gadget toolkit: Subgaussian sampling and more. In: EUROCRYPT 2019. pp. 655–684 (2019)
[24] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC 2009. pp. 169–178 (2009)
[25] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC 2008. pp. 197–206 (2008). https://doi.org/10.1145/1374376.1374407
[26] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: CRYPTO 2013. pp. 75–92 (2013)
[27] Gorbunov, S., Vaikuntanathan, V., Wee, H.: Attribute-based encryption for circuits. In: STOC 2013. pp. 545–554 (2013)
[28] Gorbunov, S., Vaikuntanathan, V., Wee, H.: Predicate encryption for circuits from lwe. In: CRYPTO 2015. pp. 503–523 (2015)
[29] Gordon, S.D., Katz, J., Vaikuntanathan, V.: A group signature scheme from lattice assumptions. In: ASIACRYPT 2010. pp. 395–412 (2010)
[30] Gür, K.D., Polyakov, Y., Rohloff, K., Ryan, G.W., Sajjadpour, H., Savaş, E.: Practical applications of improved gaussian sampling for trapdoor lattices. IEEE Transactions on Computers **68**(4), 570–584 (2018)
[31] Halevi, S., Halevi, T., Shoup, V., Stephens-Davidowitz, N.: Implementing BP-obfuscation using graph-induced encoding. In: ACM CCS 2017. pp. 783–798 (2017)
[32] Hu, Y., Jia, H.: A new gaussian sampling for trapdoor lattices with arbitrary modulus. Designs, Codes and Cryptography **87**(11), 2553–2570 (2019)
[33] Kulkarni, D., Schmidt, D., Tsui, S.K.: Eigenvalues of tridiagonal pseudo-toeplitz matrices. Linear Algebra and its Applications **297**, 63–80 (1999)
[34] Laguillaumie, F., Langlois, A., Libert, B., Stehlé, D.: Lattice-based group signatures with logarithmic signature size. In: ASIACRYPT 2013. pp. 41–61 (2013)
[35] Lai, Q., Liu, F.H., Wang, Z.: New lattice two-stage sampling technique and its applications to functional encryption–stronger security and smaller ciphertexts. In: EUROCRYPT 2021. pp. 498–527 (2021)

[36] Langlois, A., Ling, S., Nguyen, K., Wang, H.: Lattice-based group signature scheme with verifier-local revocation. In: PKC 2014. pp. 345–361 (2014)
[37] Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: EUROCRYPT 2012. pp. 700–718 (2012)
[38] Micciancio, D., Walter, M.: Gaussian sampling over the integers: Efficient, generic, constant-time. In: CRYPTO 2017. pp. 455–485 (2017)
[39] Peikert, C.: An efficient and parallel gaussian sampler for lattices. In: CRYPTO 2010. pp. 80–97 (2010)
[40] Vershynin, R.: Introduction to the non-asymptotic analysis of random matrices. arXiv preprint arXiv:1011.3027 (2010)