Article

Improved Bernoulli Sampling for Discrete Gaussian Distributions over the Integers

Shaohao Xie ^{1,2} , Shaohua Zhuang ^{1,2} and Yusong Du ^{1,2,*} ¹ School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China; xieshh6@mail2.sysu.edu.cn (S.X.); zhuangshh@mail2.sysu.edu.cn (S.Z.)² Guangdong Key Laboratory of Information Security Technology, Guangzhou 510006, China

* Correspondence: duyusong@mail.sysu.edu.cn

† Current address: Guangzhou Higher Education Mega Center, No. 132 Waihuandong Road, Guangzhou 510006, China.

Abstract: Discrete Gaussian sampling is one of the fundamental mathematical tools for lattice-based cryptography. In this paper, we revisit the Bernoulli(-type) sampling for centered discrete Gaussian distributions over the integers, which was proposed by Lucas et al. in 2013. Combining the idea of Karney's algorithm for sampling from the Bernoulli distribution $\mathcal{B}_{e^{-1/2}}$, we present an improved Bernoulli sampling algorithm. It does not require the use of floating-point arithmetic to generate a precomputed table, as the original Bernoulli sampling algorithm did. It only needs a fixed look-up table of very small size (e.g., 128 bits) that stores the binary expansion of $\ln 2$. We also propose a noncentered version of Bernoulli sampling algorithm for discrete Gaussian distributions with varying centers over the integers. It requires no floating-point arithmetic and can support centers of precision up to 52 bits. The experimental results show that our proposed algorithms have a significant improvement in the sampling efficiency as compared to other rejection algorithms.

Keywords: Discrete Gaussian sampling; discrete Gaussian distribution; exponential distribution

Citation: Xie, S.; Zhuang, S.; Du, Y. Improved Bernoulli Sampling for Discrete Gaussian Distributions over the Integers. *Mathematics* **2021**, *9*, 378. <https://doi.org/10.3390/math9040378>

Received: 21 January 2021

Accepted: 8 February 2021

Published: 13 February 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Lattice-based cryptography is one of the most widely and deeply studied post-quantum cryptography. Discrete Gaussian sampling is one of the important mathematical tools for lattice-based cryptography [1–3]. Many lattice-based cryptosystems, including their security proofs, involve discrete Gaussian sampling.

Generally, discrete Gaussian sampling is to sample from a (multivariate) discrete Gaussian distribution over an n -dimensional lattice Λ . In particular, discrete Gaussian sampling over the integers, denoted by $\text{Sample}\mathbb{Z}$, refers to the generation of random integers that conform to Gaussian distribution $\mathcal{D}_{\mathbb{Z},\sigma,\mu}$ with parameter $\sigma > 0$ and center $\mu \in \mathbb{R}$. Since $\text{Sample}\mathbb{Z}$ is much more efficient and simpler than discrete Gaussian sampling over a general lattice, many lattice-based cryptosystems, such as [4,5], only involve discrete Gaussian sampling over the integers.

The methods of sampling from a continuous Gaussian distribution are not trivially applicable for the discrete case. Therefore, it is of great significance to study discrete Gaussian sampling and design sampling algorithms, especially for $\text{Sample}\mathbb{Z}$. This issue has attracted a lot of attention in recent years.

1.1. Related Work

For the commonly used methods (techniques) for $\text{Sample}\mathbb{Z}$, including inversion sampling, the Knuth-Yao method and the rejection sampling, we can refer to [6,7]. One can also consider the convolution technique (based on the convolution-like properties of discrete Gaussian developed by Peikert in [8]) [9,10]. In addition, Saarinen developed a new binary arithmetic coding sampler for discrete Gaussians (BAC Sampling) [11]. Moreover,

Karmakar et al. designed a class of very efficient discrete Gaussian samplers based on computing a series of Boolean functions with a number of random bits as the input [12,13].

Sampling from a discrete Gaussian distribution over the integers usually involves floating-point arithmetic because of the exponential computation in Gaussian function. It is believed that high-precision floating-point arithmetic is required for attaining enough statistical distance to the target distribution. This implies that online high-precision floating-point computation may be an implementation bottleneck. Thus, we can see that most of the improved Sample $\mathbb{Z}$ algorithms manage to use high-precision floating-point arithmetic only at offline time (such as algorithms based on CDT's or Knuth-Yao) or even to avoid floating-point arithmetic (such as Karney's sampling algorithm [14]).

Recently results show that the Bernoulli sampling, introduced by Ducas et al. in [4] for centered discrete Gaussian distributions over the integers, can be effectively improved by combining the polynomial approximation technique. Specifically, Bernoulli sampling is a kind of rejection sampling. Its rejection operation can be done very efficiently by calculating an approximated polynomial [15,16]. The polynomial is predetermined; it thus supports discrete Gaussians with a varying center $\mu \in [0, 1)$ over the integers.

Generic discrete Gaussian sampling algorithms over the integers that support a varying center μ and even a varying parameter σ are also very interesting. A discrete Gaussian sampling algorithm for an n -dimensional lattice usually requires such an algorithm as its subroutine [2,8,17,18]. However, most of the algorithms for sampling $\mathcal{D}_{\mathbb{Z},\sigma,\mu}$ and their improvements are designed only for the case where center μ is fixed in advance, such as [4,9,11,19,20]. When sampling from $\mathcal{D}_{\mathbb{Z},\sigma,\mu}$ with varying $\mu \in [0, 1)$ (e.g., double-precision numbers that are sampled from a specific distribution online), except [14], the existing algorithms available rely on floating-point arithmetic, such as the algorithms in [2,15] and the sampler used in Falcon signature [5] or require a large amount of offline computation and precomputation storage, such as the algorithms proposed by Micciancio et al. [10] and by Aguilar-Melchor et al. [21] respectively. Moreover, the sampler presented by Barthe et al. in [16] is not a generic algorithm, as it supports a varying $\mu \in \mathbb{R}$ but not a varying parameters σ . One needs to determine a new approximation polynomial for each new σ .

1.2. Our Contribution

In this paper, combining the idea of Karney's algorithm for sampling the Bernoulli distribution $\mathcal{B}_{e^{-1/2}}$, we present a generalized version of Bernoulli sampling algorithm for both centered and noncentered discrete Gaussian distributions over the integers (see Algorithm 1).

Specifically, we present an improved implementation of the Bernoulli sampling algorithm for centered discrete Gaussian distributions over the integers (see Algorithm 2). We remove the precomputed table, that is required and obtained by using floating-point arithmetic in advance in the original Bernoulli sampling algorithm. Except a fixed look-up table of very small size (e.g., 128 bits) that stores the binary expansion of $\ln 2$, our proposed algorithm requires no precomputation storage and uses no floating-point arithmetic.

We also propose a noncentered Bernoulli sampling algorithm (see Algorithm 3) that supports discrete Gaussian distributions with varying centers. It uses no floating-point arithmetic, and it is suitable for sampling $\mathcal{D}_{\mathbb{Z},\sigma,\mu}$ with $\mu \in [0, 1)$ of precision up to 52 bits. We also show that our noncentered version of sampling algorithm has the same rejection rate as the original Bernoulli sampling algorithm. Furthermore, the experimental results show that our proposed algorithms have a significant improvement in the sampling efficiency as compared to other rejection algorithms for discrete Gaussian distributions over the integers [14].

Table 1 summarizes the some existing sampling algorithms for discrete Gaussian distributions over the integers and their properties in comparison to our work. Here, λ is the security parameters. The column "FPA" and the column "exp($\cdot$)" indicate if the algorithm requires to use floating-point arithmetic online and to evaluate exp($\cdot$) online

respectively. The column “varying μ ” refers to the property of being able to produce samples (online) from discrete Gaussians with varying centers rather than a specific discrete Gaussian with a fixed center. Similarly, the column “varying σ ” refers to the property of being able to produce samples (online) from discrete Gaussians with varying parameter σ .

Table 1. Comparison of sampling algorithms over the integers.

Algorithm	Memory	FPA.	$\exp(\cdot)$	Varying μ	Varying- σ
Rejection Sampling [2]	0	Yes	Yes	Yes	Yes
Inversion Sampling [8]	$O(\lambda\sigma)$	No	No	No	No
Discrete Ziggurat [22]	var.	Yes	Yes	No	No
Bernoulli Sampling [4]	$O(\lambda \log_2 \sigma)$	No	No	No	Yes
Knuth-Yao [7]	$O(\lambda\sigma)$	No	No	No	No
Karney [14]	0	No	No	Yes	Yes
Convolution-like [10]	var.	Yes	No	Yes	Yes
BAC Sampling [11]	$O(\lambda\sigma)$	No	No	No	No
Karmakar [12,13]	0	No	No	No	No
poly. approximation [15]	$O(\lambda)$	Yes	No	Yes	Yes
poly. approximation [16]	$O(\lambda)$	No	No	No	No
our Algorithm 2	$O(1)$	No	No	No	Yes
our Algorithm 3	$O(1)$	No	No	Yes	Yes

2. Materials and Methods

2.1. Preliminaries

In this paper, $\mathbb{R}$ denotes the set of real numbers, $\mathbb{Z}$ and $\mathbb{Z}^+$ denotes the set of integers and non-negative integers respectively. We use $f(S) = \sum_{x \in S} f(x)$ to define any real function $f(\cdot)$ to a countable set S if it exists, and use $\rho_{\sigma,\mu}(x) = \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$ to define the Gaussian function evaluated at $x \in \mathbb{R}$, where parameter $\sigma > 0$ and center $\mu \in \mathbb{R}$. Besides, $\mathcal{D}_{\mathbb{Z},\sigma,\mu} = \rho_{\sigma,\mu}(x)/\rho_{\sigma,\mu}(\mathbb{Z})$ defines the discrete Gaussian distribution evaluated at $x \in \mathbb{Z}$, where $x \in \mathbb{Z}$, $\mu \in \mathbb{R}$, $\sigma > 0$.

Bernoulli(-type) sampling algorithm proposed by Ducas et al. in [4] was designed exclusive for a centered discrete Gaussian distribution over the integers, namely $\mathcal{D}_{\mathbb{Z},\sigma,\mu}$ with $\sigma = k\sigma_2$ and $\mu = 0$, where k is a positive integer and $\sigma_2 = \sqrt{1/(2 \cdot \ln 2)}$. This algorithm uses rejection sampling from the *binary discrete Gaussian distribution*, denoted by $\mathcal{D}_{\mathbb{Z}^+,\sigma_2}$. Sampling from $\mathcal{D}_{\mathbb{Z}^+,\sigma_2}$ can be very efficient using only unbiased random bits (one may refer to Algorithm 10 in [4]). Firstly, it samples an integer $x \in \mathbb{Z}^+$ from $\mathcal{D}_{\mathbb{Z}^+,\sigma_2}$, then samples $y \in \mathbb{Z}$ uniformly in $\{0, 1, 2, \dots, k-1\}$ and accepts $z = kx + y$ with probability $\exp(-(y^2 + 2kxy)/2k^2\sigma_2^2)$, finally, it outputs a signed integer, z or $-z$, with equal probabilities because of the symmetry of the target distribution (see Algorithm 11 and Algorithm 12 in [4]). It was shown that the average number of rejections is not more than 1.47.

A precomputed look-up table is required for computing the exponential function $\exp(-(y^2 + 2kxy)/2\sigma^2)$. The size of the table varies with the parameter σ . For examples, the size is 2.3 kb for $\sigma \approx 107$ and 4 kb for $\sigma \approx 271$ according to the estimations given by Ducas et al. in [4], which is not very small but acceptable.

Algorithm 1 is adapted by Karney [14] from Von Neumann’s algorithm that samples from e^{-x} , where $x > 0$. Algorithm 1 is used as one of the basic operations in Karney’s sampling algorithm for the standard normal distribution, which can be executed only by using integer arithmetic.

In Algorithm 1, by replacing $1/2$ with p/q , we can get a Bernoulli random value. The value is true with probability $e^{-p/q}$, namely sampling from $\mathcal{B}_{e^{-p/q}}$, where p, q are positive integers and $p < q$. The above modification can be regarded as a generalized version of Algorithm 1, which is used implicitly in Karney’s algorithm. The discrete Gaussian distribution over integers in Karney’s algorithm also adopts the rejection sampling algorithm, which is a discretization of Karney’s algorithm for sampling from the standard normal

distribution. It is fit for discrete Gaussians with varying rational parameters of precision up to 32 bits (including σ and μ). The algorithm does not require any floating-point arithmetic. Moreover, it needs no precomputation storage. One may refer to [14] for the full description of Karney's sampling algorithm.

Algorithm 1: [14] Sample from Bernoulli distribution $\mathcal{B}_{1/\sqrt{e}}$

Output: a Bernoulli random value from $\mathcal{B}_{1/\sqrt{e}}$

```

1 sample uniform deviates  $u_1, u_2, \dots$  with  $u_i \in [0, 1)$  and determine the maximum
  value  $n \geq 0$  such that  $1/2 > u_1 > u_2 > \dots > u_n$ ;
2 if  $n$  is even then
3   | return true;
4 else
5   | return false;
```

2.2. Removing Precomputed Tables in Bernoulli Sampling

One drawback of the Bernoulli sampling is that it needs a precomputed table to generate a Bernoulli random value from $\mathcal{B}_{\exp(-(y^2+2kxy)/2\sigma^2)}$. Although the table is not large, it has to be precomputed before sampling for every different $\sigma = k\sigma_2$. Therefore, it is interesting to sample $\mathcal{D}_{\mathbb{Z}, k\sigma_2}$ without a precomputed table. In this subsection, we propose an alternative algorithm, which samples $\mathcal{D}_{\mathbb{Z}, k\sigma_2}$ for a given $k \geq 1$ and does not require a precomputed table except a fixed look-up table with small size.

We use the notation of the sampling algorithm given by Lucas et al. (Algorithm 11 in [4]). Since $\sigma_2 = \sqrt{1/(2 \cdot \ln 2)}$ and $\sigma = k\sigma_2$, it follows that $\exp(-(y^2 + 2kxy)/2\sigma^2)$ is equal to

$$\begin{aligned} & \exp\left(-\ln 2 \left(\left\lfloor \frac{y^2 + 2kxy}{k^2} \right\rfloor + \left\{ \frac{y^2 + 2kxy}{k^2} \right\}\right)\right) \\ &= \exp\left(-\ln 2 \left\lfloor \frac{y^2 + 2kxy}{k^2} \right\rfloor\right) \cdot \exp\left(-\ln 2 \left\{ \frac{y^2 + 2kxy}{k^2} \right\}\right) \end{aligned}$$

where

$$\left\{ \frac{y^2 + 2kxy}{k^2} \right\} = \frac{y^2 + 2kxy}{k^2} - \left\lfloor \frac{y^2 + 2kxy}{k^2} \right\rfloor = \frac{(y^2 + 2kxy) \bmod k^2}{k^2}$$

is the fractional part of $(y^2 + 2kxy)/k^2$. Then the Bernoulli value can be obtained by generating $\lfloor y^2 + 2kxy/k^2 \rfloor$ Bernoulli deviates according to $\mathcal{B}_{1/2}$ and one Bernoulli random value from $\mathcal{B}_{\exp(-(\ln 2)\{y^2+2kxy/k^2\})}$. If no false value is generated in this procedure, then we have **true**, otherwise we have **false**.

It is clear that a Bernoulli deviate according to $\mathcal{B}_{1/2}$ is equivalent to a uniform random bit. Thus, the only remaining problem is to generate a Bernoulli random value from $\mathcal{B}_{\exp(-(\ln 2)\{y^2+2kxy/k^2\})}$. We address this problem via Algorithm 2.

One can see that Algorithm 2 is adapted from Algorithm 1. It provides a technique for determining the relation between $(\ln 2)(z/k^2)$ and a uniform deviate $u_1 \in [0, 1)$, and it does not need floating-point operations during run-time, if the binary expansion of $\ln 2$, that is long enough (e.g., 128 bits), has been stored as a fixed look-up table.

In step 1, p is assigned to be the t most significant bits of the binary expansion of $\ln 2$. In steps 5 and 6, if $\lfloor p \rfloor > \lfloor q/z \rfloor = \lfloor (vk^2)/z \rfloor$, then we have $2^t(\ln 2) > (vk^2)/z$, which follows that $(\ln 2)(z/k^2) > v/2^t$. It is equivalent to obtaining a uniform deviate $u_1 = (v/2^t)$ that is strictly less than the value of $(\ln 2)(z/k^2)$. On the contrary, in steps 7 and 8, if $\lfloor p \rfloor < \lfloor q/z \rfloor$, then it is equivalent to obtaining a uniform deviate u_1 that is strictly more than the value of $(\ln 2)(z/k^2)$. Otherwise, if $\lfloor p \rfloor = \lfloor q/z \rfloor$, i.e., $\lfloor 2^t(\ln 2) \rfloor = \lfloor (vk^2)/z \rfloor$. This means that we get a uniform deviate u_1 , but in this moment we cannot determine whether it is less than the value of $(\ln 2)(z/k^2)$ or not.

Algorithm 2: Sample from Bernoulli distribution $\mathcal{B}_{\exp(-(\ln 2)(z/k^2))}$, where k, z are positive integers such that $z < k^2$

Input: positive integers k, z, t such that $z < k^2$ and $t \geq 1$
Output: a Bernoulli random value from $\mathcal{B}_{\exp(-(\ln 2)(z/k^2))}$

```

1 set  $u_1 \leftarrow 0, i \leftarrow 0$  and  $p \leftarrow (\ln 2) \cdot 2^t$ ;
2 sample a uniform integer  $v \in [0, 2^t)$  and set  $q \leftarrow vk^2$ ;
3 while true do
4   set  $i \leftarrow i + 1, u_1 \leftarrow u_1 + v/2^{it}$ ;
5   if  $\lfloor p \rfloor > \lfloor q/z \rfloor$  then
6     break
7   if  $\lfloor p \rfloor < \lfloor q/z \rfloor$  then
8     return true;
9   set  $p \leftarrow (p - \lfloor p \rfloor) \cdot 2^t$  and  $q \leftarrow q \pmod{z}$ ;
10  update the uniform integer  $v \in [0, 2^t)$ ;
11  set  $q \leftarrow (q \cdot 2^t + vk^2)$ ;
12 sample uniform deviates  $u_2, u_3 \dots$  with  $u_i \in [0, 1]$  and determine the maximum
    value  $n \geq 1$  such that  $u_1 > u_2 > u_3 > \dots > u_n$ ;
13 if  $n$  is even then
14   return true;
15 else
16   return false;

```

In Algorithm 2, the while loop is designed to further compare the uniform deviate with the value of $(\ln 2)(z/k^2)$. Informally, if $\lfloor 2^t(\ln 2) \rfloor = \lfloor (vk^2)/z \rfloor$ for some $v_1 \in [0, 2^t)$, then we generate another uniformly random integer $v_2 \in [0, 2^t)$, which means that we get a uniform deviate $u_1 \geq (v_1/2^t) + (v_2/2^{2t})$. When comparing $(\ln 2)(z/k^2)$ with $(v_1/2^t) + (v_2/2^{2t})$, if

$$2^t(\ln 2) > \frac{v_1 k^2}{z} + \frac{v_2 k^2}{z 2^t}$$

then we have $(\ln 2)(z/k^2) > (v_1/2^t) + (v_2/2^{2t})$, which follows that $(\ln 2)(z/k^2) > u_1$. Since $\lfloor 2^t(\ln 2) \rfloor = \lfloor (vk^2)/z \rfloor$, we have $(\ln 2)(z/k^2) > u_1$ if

$$\lfloor 2^t(\ln 2) \rfloor + \{2^t(\ln 2)\} > \lfloor \frac{v_1 k^2}{z} \rfloor + \{\frac{v_1 k^2}{z}\} + \frac{v_2 k^2}{z 2^t},$$

i.e.,

$$\{2^t(\ln 2)\} > \{\frac{v_1 k^2}{z}\} + \frac{v_2 k^2}{z 2^t},$$

which is equivalent to

$$2^t\{2^t(\ln 2)\} > 2^t\{\frac{v_1 k^2}{z}\} + \frac{v_2 k^2}{z},$$

where $\{v_1 k^2/z\}$ is the fractional part of the rational number $v_1 k^2/z$, which can be determined by computing $v_1 k^2 \pmod{z}$. Therefore, as presented in Algorithm 2, after steps 9, 10, and 11 what we get is actually $p = (p - \lfloor p \rfloor) \cdot 2^t = (2^t(\ln 2) - \lfloor 2^t(\ln 2) \rfloor)2^t = 2^t\{2^t(\ln 2)\}$ and $q = ((v_1 k^2 \pmod{z}) \cdot 2^t + v_2 k^2)$. So, we can continue to compare the uniform deviate u_1 with the value of $(\ln 2)(z/k^2)$. For the full proof of the correctness, one may refer to Appendix A.

2.3. Bernoulli Sampling for Discrete Gaussians with Varying Centers

In this subsection, we propose a new noncentered Bernoulli sampling algorithm for $\mathcal{D}_{\mathbb{Z}, \sigma, i/b}$ with $\sigma = k\sigma_2$, where b is a power of two, $i \in [0, b-1] \cap \mathbb{Z}$, k is a positive integer, $\sigma_2 = \sqrt{1/(2 \cdot \ln 2)}$. Our proposed algorithm does not require any floating-point arithmetic,

and it needs no precomputed table except a fixed and small size look-up table for storing the binary expansion of $\ln 2$, as mentioned in Section 2.2.

Informally, the sketch of the proposed algorithm is as follows. We sample an integer $x \in \mathbb{Z}^+$ from $\mathcal{D}_{\mathbb{Z}^+, \sigma_2}$ firstly, and then sample $y \in \mathbb{Z}$ uniformly in $\{0, 1, 2, \dots, k\}$. Since the symmetry of a noncentered discrete Gaussian distribution does not hold, we need to determine the sign of prospective output before the acceptance-rejection operation, by setting $s \leftarrow \pm 1$ with equal probabilities. Finally, we accept $s(kx + y)$ as the output with probability

$$\exp\left(-\frac{(y - s\mu)^2 + 2kx(y - s\mu)}{2k^2\sigma_2^2}\right),$$

and restart the whole algorithm if otherwise, where $\mu = i/b$. The following equation

$$\exp\left(-\frac{x^2}{2\sigma_2^2}\right) \cdot \exp\left(-\frac{(y - s\mu)^2 + 2kx(y - s\mu)}{2k^2\sigma_2^2}\right) = \exp\left(-\frac{(s(kx + y) - \mu)^2}{2k^2\sigma_2^2}\right)$$

guarantees that the output value has the desired relative probability density $\rho_{k\sigma_2, \mu}(s(kx + y))$. The sketch of the algorithm seems to work, but there are several subtleties we have to address.

Bernoulli sampling for centered discrete Gaussian distributions needs to restart its whole sampling procedure with probability $1/2$ when the algorithm is about to return $z = 0$ (see Algorithm 12 in [4]). In our algorithm, in order to avoid double counting 0, we need to set $y \leftarrow y + 1$ if $s = 1$, so that there is only one case where $z = 0$. This also guarantees that $y - s\mu$ is always non-negative, and thus $\exp(-(y - s\mu)^2 + 2kx(y - s\mu))/(2k^2\sigma_2^2)$ is never more than one.

Another problem is how to compute the value of the exponential function efficiently. Since $\sigma_2 = \sqrt{1/(2 \cdot \ln 2)}$ and $\mu = i/b$ with $i \in [0, b - 1] \cap \mathbb{Z}$, it follows that

$$\exp\left(-\frac{(y - s\mu)^2 + 2kx(y - s\mu)}{2k^2\sigma_2^2}\right) = 2^{-\lfloor \delta \rfloor} + 2^{-\{\delta\}}$$

where $\delta = ((by - si)^2 + 2bkx(by - si))/(b^2k^2)$ and

$$\{\delta\} = \delta - \lfloor \delta \rfloor = \frac{((by - si)^2 + 2bkx(by - si)) \bmod (b^2k^2)}{b^2k^2}$$

is the fractional part of δ . Then, when computing the value of exponential function $\exp(-(y - s\mu)^2 + 2kx(y - s\mu))/(2k^2\sigma_2^2)$, we generate $\lfloor \delta \rfloor$ Bernoulli deviates according to $\mathcal{B}_{1/2}$ and one Bernoulli random value which is true with probability $\exp(-\ln 2 \cdot \{\delta\}) = 2^{-\{\delta\}}$. If no false value is generated, then we accept $s(kx + y)$ as the output, otherwise we restart the algorithm. In summary, our proposed algorithm can be formally described as Algorithm 3, which summarizes the above discussion.

Recall that our goal is to give an algorithm that requires no floating-point arithmetic. It is clear that a Bernoulli deviate according to $\mathcal{B}_{1/2}$ is equivalent to a uniform random bit, and sampling from $\mathcal{D}_{\mathbb{Z}^+, \sigma_2}$ can be efficiently implemented by using only unbiased random bits (Algorithm 10 in [4]). Thus, the only remaining problem is to generate a Bernoulli random value with probability

$$\exp(-\ln 2 \cdot \{\delta\}) = \exp\left(-\ln 2 \cdot \left\{\frac{(by - si)^2 + 2bkx(by - si)}{b^2k^2}\right\}\right)$$

without using floating-point arithmetic. Since $(by - si)^2 + 2bkx(by - si)$ is a non-negative integer, what we need essentially is a solution for generating a Bernoulli random value from $\mathcal{B}_{\exp(-(\ln 2)(z/b^2k^2))}$, where $z = (by - si)^2 + 2bkx(by - si)$ and k are positive integers

such that $z < b^2k^2$. To end this, we can apply Algorithm 2 by replacing k with b^2k^2 , which involves no floating-point arithmetic and requires no precomputation storage.

Algorithm 3: Sample from discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z},k\sigma_2,i/b}$ for $k \in \mathbb{Z}^+$, $0 \leq i < b$ and $b \geq 2$

Input: k, i and b
Output: an integer z according to $\mathcal{D}_{\mathbb{Z},k\sigma_2,i/b}$

- 1 sample $x \in \mathbb{Z}$ according to $\mathcal{D}_{\mathbb{Z}^+, \sigma_2}$;
- 2 sample $y \in \mathbb{Z}$ uniformly in $\{0, 1, 2, \dots, k-1\}$;
- 3 set $s \leftarrow \pm 1$ with equal probabilities;
- 4 **if** $s = 1$ **then**
- 5 | set $y \leftarrow y + 1$;
- 6 set $z \leftarrow s(kx + y)$ and $\delta = ((by - si)^2 + 2bkx(by - si))/b^2k^2$;
- 7 sample a Bernoulli random value β according to $\mathcal{B}_{\exp(-(\ln 2) \cdot (\delta - \lfloor \delta \rfloor))}$;
- 8 **if** $\beta = \text{true}$ **then**
- 9 | **goto** step 1 and **restart**;
- 10 sample $\lfloor \delta \rfloor$ Bernoulli random values according to $\mathcal{B}_{1/2}$;
- 11 **if all** $\lfloor \delta \rfloor$ Bernoulli values are **true** **then**
- 12 | **return** z ;
- 13 **else**
- 14 | **goto** step 1 and **restart**;

Moreover, let $k < 2^8$ and $b = 2^{20}$, then we have $(by - si)^2 \leq b^2k^2 < 2^{56}$ and $2bkx(by - si) < 2^{60}$ with overwhelming probability. This means that Algorithm 3 supports a μ of precision up to 20 bits if k is restricted to be less than 2^8 , and it can be implemented only by using 64-bit integer arithmetic.

A sampler for centers of low precision can be used in q -ary lattices, where q is relatively large but not very large, and sampling from discrete Gaussian distributions with q different centers may be required in this case. It involves a large storage requirement if one uses precomputed tables. So, one could consider our sampler as described in Algorithm 3. It requires no precomputed table.

2.4. Increasing the Precision of μ

Algorithm 3 only supports centers of finite precision. The precision of μ depends on the value of b . A larger b means a μ of higher precision. For example, let $b = 2^{52}$. We remark that the bound of 52 bits is significant. We note that float-pointing with 53 bits of precision corresponds to the double precision type in the IEEE 754 standard, and it is usable in software. When $\mu \in [0, 1)$ is given by a double-precision floating-point number, we could represent μ as a rational number of denominator 2^{52} and use Algorithm 3 directly.

However, b cannot be 2^{52} in practice, since in this case we need to compute

$$\frac{(by - si)^2 + 2bkx(by - si)}{b^2k^2},$$

where $b^2k^2 > 2^{64}$, which leads to the integer overflow (for 64-bit C/C++ compilers). A big integer library works but compromises the performance at runtime. In this subsection, therefore, we try to improve Algorithm 3 so that it supports a μ of precision up to 52 bits.

The basic idea is to avoid compute b^2k^2 . We note that generating a Bernoulli random value which is true with probability

$$\exp\left(-\ln 2 \left(\frac{(by - si)^2 + 2bkx(by - si)}{b^2k^2}\right)\right)$$

can be decomposed to generating two Bernoulli random values which are true with probability $\exp\left(-\ln 2\left(\frac{(by-si)}{bk}\right)^2\right)$ and $\exp\left(-\ln 2\left(\frac{2x(by-si)}{bk}\right)\right)$ respectively. It is not hard to see that a Bernoulli random value which is true with probability $\exp\left(-\ln 2\left(\frac{2x(by-si)}{bk}\right)\right)$ can be generated by using Algorithm 2 with $z = 2x(by - si)$ and $k^2 = bk$. In particular, Algorithm 2 requires $2x(by - si) < bk$. When $2x(by - si) > bk$, we need to compute $\lfloor 2x(by - si)/bk \rfloor$ and invoke Algorithm 1 firstly, and then use Algorithm 2 with $z = 2x(by - si) \bmod (bk)$ and $k^2 = bk$.

The following algorithm is designed to generate a Bernoulli random value from Bernoulli distribution $\mathcal{B}_\theta$ with $\theta = \exp\left(-\ln 2\left(\frac{z}{K}\right)^2\right)$. As the inequality $by - si < bk$ holds except $i = 0$ and $y = 0$, $\exp\left(-\ln 2\left(\frac{(by-si)}{bk}\right)^2\right)$ can be calculated by using Algorithm 4 with $z = (by - si)$ and $K = bk$.

Algorithm 4: Sample from Bernoulli distribution $\mathcal{B}_\theta$ with $\theta = \exp\left(-\ln 2\left(\frac{z}{K}\right)^2\right)$ and $0 < z < K$

Input: integer K and integer z such that $0 < z < K$

Output: a Bernoulli random value from $\mathcal{B}_\theta$

```

1 set  $x \leftarrow (z/K) \cdot \ln 2$  and  $n \leftarrow 0$ ;
2 sample uniform deviates  $u$  with  $u \in [0, 1)$ ;
3 while  $u < x$  do
4   sample uniform deviates  $v$  with  $v \in [0, 1)$ ;
5   if  $v < (z/K)$  then
6     set  $x \leftarrow u$ ,  $n \leftarrow n + 1$ ;
7     sample uniform deviates  $u$  with  $u \in [0, 1)$ ;
8   else
9     break and goto step 10;
10 if  $n$  is even then
11   return true;
12 else
13   return false;
```

The main idea behind Algorithm 4 is to sample two sets of uniform deviates $u_1, u_2, \dots$ and $v_1, v_2, \dots$, and to decide the maximum value $n \geq 0$ such that $\ln 2 \cdot (z/K) > u_1 > u_2 > \dots > u_n$ and $v_i < z/k$. Then, the probability that n is even, which means it returns **true**, is given by

$$\begin{aligned}
& 1 - \frac{((z/K) \cdot \ln 2)}{1!} \left(\frac{z}{K}\right) + \frac{((z/K) \cdot \ln 2)^2}{2!} \left(\frac{z}{K}\right)^2 - \frac{((z/K) \cdot \ln 2)^3}{3!} \left(\frac{z}{K}\right)^3 + \dots \\
& = 1 - \frac{\ln 2}{1!} \left(\frac{z}{K}\right)^2 + \frac{(\ln 2)^2}{2!} \left(\frac{z}{K}\right)^4 - \frac{(\ln 2)^3}{3!} \left(\frac{z}{K}\right)^6 + \dots = \exp\left(-(\ln 2) \left(\frac{z}{K}\right)^2\right).
\end{aligned}$$

Furthermore, when implementing Algorithm 4, we need to apply the technique presented in Algorithm 2 for comparing u with $(z/K) \cdot \ln 2$ and avoiding using floating-point arithmetic.

Let us go back to the problem of increasing the precision of μ . If k is restricted to be less than 2^8 , we can take $b = 2^{52}$. In this case, we have $(by - si) \leq bk < 2^{60}$ and $2x(by - si) < 2^{64}$ with overwhelming probability. This means that Algorithm 3, incorporated with Algorithm 4 as well as Algorithm 2, supports a μ of precision up to 52 bits if k is restricted to be less than 2^8 , and it can be implemented only by using 64-bit integer arithmetic.

3. Results

On a laptop computer (Intel i7-8550U, 16 GB RAM, the g++ compiler and enabling -O3 optimization option), we implemented GPV algorithm in [2], Karney's algorithm in [14], the original Bernoulli sampling algorithm in [4], and our Algorithm 3 respectively. Except Karney's algorithm, the uniformly pseudorandom numbers are generated by using AES256-CTR with AES-NI (Intel Advanced Encryption Standard New Instructions). Figure 1 shows the performance comparison of these sampling algorithms.

Figure 1. Performance Comparison of Sampling Algorithms.

We select Karney's algorithm and the GPV algorithm to compare with our generalized Bernoulli sampling because they also use rejection sampling and requires no precomputation storage. Although algorithms based on the inversion sampling or Knuth-Yao method are usually more efficient, they require a large amount precomputation storage, especially for sampling from distributions with variable μ or large σ .

For centered discrete Gaussian distributions with σ roughly from 10 to 190, one can get about 12.7×10^6 samples per second by using Algorithm 3. This means that the performance of our improved Bernoulli sampling algorithm is significantly better than the original Bernoulli sampling as well as Karney's sampling algorithm. When $b = 2^{20}$, for the discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}, \sigma, \mu}$ with $\mu = i/b$ and i uniformly from $[0, b-1]$, we also tested that Algorithm 3 has almost the same performance as the case of centered discrete Gaussian distributions. When we take $b = 2^{52}$, we need combine Algorithm 4 to implement Algorithm 3. The performance in this case decreases to about 6.4×10^6 per second and is slightly worse than the original Bernoulli sampling algorithm, but the latter does not support noncentered discrete Gaussian distributions.

Here, the GPV algorithm has the worst performance, since its proposal distribution is poor and its rejection rate is too high, which is close to 10 [2], while the Bernoulli sampling algorithm has a rejection rate of about 1.47, and the rejection rate of Karney's algorithm is even as lower as 1.39 [14]. We will prove in Section 4 of this paper that the rejection rate of Algorithm 3 is still about 1.47.

Our Algorithm 3 has a better performance than Karney's algorithm, this is due to the fact that it uses the binary discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}, \sigma_2}$ as its base sampler instead of $\mathcal{D}_{\mathbb{Z}, 1}$, and there exists a very efficient sampling algorithm for such a discrete Gaussian distribution. Algorithm 3 is faster than the original Bernoulli sampling algorithm, mainly because its rejection operation is simplified to sample from only one Bernoulli distribution with an exponential function as its parameter, while the original Bernoulli sampling algorithm requires a lookup table to sample a relatively large number of Bernoulli distributions with exponential functions as parameters in order to perform the rejection operation, which affects its practical performance.

Note that our Algorithm 4 requires Algorithm 2 to be fully implemented, and the basic idea of Algorithm 4 follows from that of Algorithm 2. We perform algorithm 4 once roughly as the same amount of computation as Algorithm 2 twice. This helps us intuitively explain why the performance of Algorithm 3 drops by about half after using Algorithm 4.

4. Discussion

In this section, we show that the noncentered version of Bernoulli sampling presented by Algorithm 3 has the same rejection rate as the Bernoulli sampling for centered discrete Gaussians.

Intuitively, evaluating the rejection rate of Algorithm 3 could be done just as evaluating the rejection rate of Bernoulli sampling for centered discrete Gaussian distributions, which has been presented by Ducas et al. in [4]. Notwithstanding, we may need the notion of the smoothing parameter of lattices, which was introduced by Micciancio and Regev in [1], because of the noncentered discrete Gaussian distributions. In fact, we only need the notion of smoothing parameter of lattice $\mathbb{Z}$ and its two important properties, since this work only involves the (one-dimensional) integer lattice $\mathbb{Z}$, which is the simplest lattice.

Definition 1 (a special case of [1], Definition 3.1). *Let $\epsilon > 0$ be a positive real. The smoothing parameter of lattice $\mathbb{Z}$, denoted by $\eta_\epsilon(\mathbb{Z})$, is defined to be the smallest real s such that $\rho_{1/s}(\mathbb{Z} \setminus \{0\}) \leq \epsilon$.*

Lemma 1 (a special case of [1], Lemma 3.3). *For any real $\epsilon > 0$, the smoothing parameter of lattice $\mathbb{Z}$ satisfies $\eta_\epsilon(\mathbb{Z}) \leq \sqrt{\ln(2(1 + 1/\epsilon))}/2/\pi$.*

Lemma 2 (Ref. [1], Lemma 2.8, implicit). *For any (small) real $\epsilon > 0$ and $\mu \in \mathbb{R}$, if $\sigma \geq \eta_\epsilon(\mathbb{Z})$, then $\rho_{\sigma,\mu}(\mathbb{Z}) = \left[\frac{1-\epsilon}{1+\epsilon}, 1\right] \cdot \rho_\sigma(\mathbb{Z})$, and $\rho_\sigma(\mathbb{Z}) \approx \int_{-\infty}^{+\infty} \rho_\sigma(x) dx = \sqrt{2\pi}\sigma$.*

Theorem 1. *For Algorithm 3, if $\sigma = k\sigma_2 \geq \eta_\epsilon(\mathbb{Z})$, then the expected number of rejections that it requires is not more than $\rho_{\sigma_2}(\mathbb{Z}^+)/\sigma_2\sqrt{\pi}/2 \approx 1.47$.*

Proof. From the perspective of rejection sampling, for Algorithm 3, the proposal distribution density function is $g(z) = \rho_{\sigma_2}(x)/(2k\rho_{\sigma_2}(\mathbb{Z}^+))$ and the target distribution density function f can be written as

$$f(z) = f(s(kx + y + (1 + s)/2)) = \frac{\rho_{k\sigma_2,\mu}(s(kx + y + (1 + s)/2))}{\rho_{k\sigma_2,\mu}(\mathbb{Z})},$$

where $s \in \{0, 1\}$, $\mu = i/b$ and $i \in [0, b/2] \cap \mathbb{Z}$. The expected number of rejections that the algorithm requires can be evaluated by

$$\max \frac{f(z)}{g(z)} = \max \left(\frac{\rho_{k\sigma_2,\mu}(s(kx + y + (1 + s)/2))}{\rho_{\sigma_2}(x)} \cdot \frac{2k\rho_{\sigma_2}(\mathbb{Z}^+)}{\rho_{k\sigma_2,\mu}(\mathbb{Z})} \right) \leq \frac{2k\rho_{\sigma_2}(\mathbb{Z}^+)}{\rho_{k\sigma_2,\mu}(\mathbb{Z})}.$$

If $\sigma = k\sigma_2 \geq \eta_\epsilon(\mathbb{Z})$, then $\rho_{k\sigma_2,\mu}(\mathbb{Z}) \approx \rho_{k\sigma_2}(\mathbb{Z})$ (or $\rho_{k\sigma_2,\mu}(\mathbb{Z}) = \rho_{k\sigma_2}(\mathbb{Z})$ when $\mu = 0$). Then we have

$$\frac{2k\rho_{\sigma_2}(\mathbb{Z}^+)}{\rho_{k\sigma_2,\mu}(\mathbb{Z})} \approx \frac{2k\rho_{\sigma_2}(\mathbb{Z}^+)}{\rho_{k\sigma_2}(\mathbb{Z})} \approx \frac{2k\rho_{\sigma_2}(\mathbb{Z}^+)}{2k\sigma_2\sqrt{\pi}/2} = \frac{\rho_{\sigma_2}(\mathbb{Z}^+)}{\sigma_2\sqrt{\pi}/2} \approx 1.47,$$

where the two approximate equality symbols follow from Lemma 2. This completes the proof. $\square$

We remark that Theorem 1 holds for $k \geq 1$ in practice. In the proof of Theorem 1, taking $\epsilon = 2^{-19}$ we have $(1 - \epsilon)/(1 + \epsilon) \approx 1$, which guarantees that $\rho_{\sigma,\mu}(\mathbb{Z})$ gets close enough to $\rho_\sigma(\mathbb{Z})$ by Lemma 2. According to Lemma 1, we get $\eta_\epsilon(\mathbb{Z}) \leq \sqrt{\ln(2(1 + 1/\epsilon))}/2/\pi \approx 0.838036$ with $\epsilon = 2^{-19}$.

We can also see that the rejection rate of 1.47 also represents the average number of times that Algorithm 3 calls Algorithm 2 or Algorithm 4. This is because the rejection rate of Algorithm 3 is actually equal to the inverse of the probability that the algorithm executes step 7 and step 10 without restarting. Algorithm 3 needs to perform step 7 whenever it successfully outputs the sample or restarts, and step 7 requires a call to Algorithm 2 or

Algorithm 4. Thus, the average number of times that step 7 is executed is the rejection rate, and the average number of calls of Algorithm 2 or Algorithm 4 is thus the rejection rate, which is about 1.47 and independent from the parameter σ and the center μ , according to Theorem 1.

5. Conclusions

In this paper, we show that the Bernoulli(-type) sampling for centered discrete Gaussian distributions over the integers can be implemented without floating-point arithmetic. Except a fixed look-up table of very small size (e.g., 128 bits) that stores the binary expansion of $\ln 2$, our proposed algorithm does not need any precomputation storage and requires no floating-point arithmetic. We also present a noncentered Bernoulli sampling algorithm that satisfies the need of discrete Gaussian distributions with varying centers. The algorithm needs no floating-point arithmetic, and it is suitable for sampling $\mathcal{D}_{\mathbb{Z}, \sigma, \mu}$ with $\mu \in [0, 1)$ of precision up to 52 bits. The noncentered version of algorithm has the same rejection rate as the original Bernoulli sampling algorithm. The experiment shows that our improved algorithms have better performance as compared to other rejection algorithms for discrete Gaussian distributions over the integers.

Author Contributions: Conceptualization, Y.D.; methodology, Y.D.; software, Y.D.; validation, S.X. and S.Z.; formal analysis, Y.D. and S.Z.; investigation, S.X. and S.Z.; resources, S.X.; data curation, S.X.; writing—original draft preparation, Y.D.; writing—review and editing, S.X.; visualization, S.X. and S.Z.; supervision, Y.D.; project administration, Y.D.; funding acquisition, Y.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Natural Science Foundations of China (Grant Nos. 61972431), Guangdong Major Project of Basic and Applied Research (2019B030302008), Guangdong Basic and Applied Basic Research Foundation (No. 2020A1515010687) and the Fundamental Research Funds for the Central Universities (Grant No. 19lgpy217).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A

Let $v_1, v_2, \dots, v_n$ be n integers such that $v_i \in [0, 2^t)$ for each $i \geq 1$. For $i \geq 2$, $G_t(i)$ and $H_t(k, z; v_1, v_2, \dots, v_i)$ are defined as follows:

$$G_t(i) = 2^t \{G_t(i-1)\} = \underbrace{2^t \{ \dots 2^t \{2^t(\ln 2)\} \dots \}}_i$$

and

$$\begin{aligned} H_t(k, z; v_1, \dots, v_i) &= 2^t \{H_t(k, z; v_1, \dots, v_{i-1})\} + \frac{v_i k^2}{z} \\ &= \underbrace{2^t \{ \dots 2^t \{2^t \{ \frac{v_1 k^2}{z} \} + \frac{v_2 k^2}{z} \} \dots \}}_{i-1} + \frac{v_i k^2}{z}. \end{aligned}$$

In particular, $G_t(1) = 2^t(\ln 2)$ and $H_t(k, z; v_1) = v_1 k^2 / z$.

Theorem A1. Let $u_1 = \sum_{i=1}^n (v_i / 2^{it}) \in [0, 1]$. Suppose that

$$\lfloor G_t(i) \rfloor = \lfloor H_t(k, z; v_1, v_2, \dots, v_i) \rfloor$$

for each integer $i = 1, 2, \dots, n-1$. If $\lfloor G_t(n) \rfloor > \lfloor H_t(k, z; v_1, v_2, \dots, v_n) \rfloor$, then $u_1 < (\ln 2)(z/k^2)$.
If $\lfloor G_t(n) \rfloor < \lfloor H_t(k, z; v_1, v_2, \dots, v_n) \rfloor$, then $u_1 > (\ln 2)(z/k^2)$.

Proof. Since $\lfloor G_t(n) \rfloor > \lfloor H_t(k, z; v_1, v_2, \dots, v_n) \rfloor$, we have

$$G_t(n) > H_t(k, z; v_1, v_2, \dots, v_n).$$

It follows that $\{G_t(n-1)\} = G_t(n)/2^t$ and

$$\frac{G_t(n)}{2^t} > \frac{H_t(k, z; v_1, v_2, \dots, v_n)}{2^t} = \{H_t(k, z; v_1, v_2, \dots, v_{n-1})\} + \frac{v_n k^2}{2^t z}.$$

Then, we have

$$\begin{aligned} & \{G_t(n-1)\} + \lfloor G_t(n-1) \rfloor \\ & > \{H_t(k, z; v_1, v_2, \dots, v_n)\} + \lfloor H_t(k, z; v_1, v_2, \dots, v_{n-1}) \rfloor + \frac{v_n k^2}{2^t z}, \end{aligned}$$

as $\lfloor G_t(n-1) \rfloor = \lfloor H_t(k, z; v_1, v_2, \dots, v_{n-1}) \rfloor$. Thus, we have

$$G_t(n-1) > H_t(k, z; v_1, v_2, \dots, v_{n-1}) + \frac{v_n k^2}{2^t z}.$$

If $n-1 > 1$, with similar arguments, applying $\lfloor G_t(i) \rfloor = \lfloor H_t(k, z; v_1, v_2, \dots, v_i) \rfloor$ with $i = n-2$, we get

$$G_t(n-2) > H_t(k, z; v_1, v_2, \dots, v_{n-2}) + \frac{v_{n-1} k^2}{2^t z} + \frac{v_n k^2}{2^{2t} z}.$$

We continue working backwards until finally we obtain

$$G_t(1) > H_t(k, z; v_1) + \frac{v_2 k^2}{2^t z} + \frac{v_3 k^2}{2^{2t} z} + \dots + \frac{v_n k^2}{2^{(n-1)t} z},$$

which means that

$$2^t (\ln 2) > \frac{v_1 k^2}{z} + \frac{v_2 k^2}{2^t z} + \frac{v_3 k^2}{2^{2t} z} + \dots + \frac{v_n k^2}{2^{(n-1)t} z}.$$

It follows that

$$\frac{(\ln 2)z}{k^2} > \frac{v_1}{2^t} + \frac{v_2}{2^{2t}} + \frac{v_3}{2^{3t}} + \dots + \frac{v_n}{2^{nt}}.$$

This is equivalent to obtaining a uniform deviate

$$u_1 = \frac{v_1}{2^t} + \frac{v_2}{2^{2t}} + \frac{v_3}{2^{3t}} + \dots + \frac{v_n}{2^{nt}} \in [0, 1],$$

which is strictly less than the value of $(\ln 2)(z/k^2)$.

Similarly, if $\lfloor G_t(i) \rfloor = \lfloor H_t(k, z; v_1, v_2, \dots, v_i) \rfloor$ for each integer $i = 1, 2, \dots, n-1$, but

$$\lfloor G_t(n) \rfloor < \lfloor H_t(k, z; v_1, v_2, \dots, v_n) \rfloor.$$

Then, we essentially get a uniform deviate u_1 that is strictly more than the value of $(\ln 2)(z/k^2)$. $\square$

References

1. Micciancio, D.; Regev, O. Worst-Case to Average-Case Reductions Based on Gaussian Measures. *SIAM J. Comput.* **2007**, *37*, 267–302. [\[CrossRef\]](#)
2. Gentry, C.; Peikert, C.; Vaikuntanathan, V. Trapdoors for hard lattices and new cryptographic constructions. In Proceedings of the STOC 2008, Victoria, BC, Canada, 17–20 May 2008; Dwork, C., Ed.; ACM: New York, NY, USA, 2008; pp. 197–206.
3. Lyubashevsky, V.; Peikert, C.; Regev, O. On Ideal Lattices and Learning with Errors over Rings. *J. ACM* **2013**, *60*, 43:1–43:35. [\[CrossRef\]](#)
4. Ducas, L.; Durmus, A.; Lepoint, T.; Lyubashevsky, V. Lattice Signatures and Bimodal Gaussians. In Proceedings of the CRYPTO 2013, Santa Barbara, CA, USA, 18–22 August 2013; Canetti, R.; Garay, J.A., Eds.; Proceedings, Part I; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8042, pp. 40–56.
5. Prest, T.; Fouque, P.A.; Hoffstein, J.; Kirchner, P.; Lyubashevsky, V.; Pornin, T.; Ricosset, T.; Seiler, G.; Whyte, W.; Zhang, Z. Falcon: Fast-Fourier-Lattice-based Compact Signatures over NTRU. Technical Report. Specifications v1.1. Available online: <https://falcon-sign.info/> (accessed on 8 January 2021).
6. Follth, J. Gaussian Sampling in Lattice Based Cryptography. *Tatra Mt. Math. Publ.* **2014**, *60*, 1–23. [\[CrossRef\]](#)
7. Dwarakanath, N.C.; Galbraith, S.D. Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. *Appl. Algebra Eng. Commun. Comput.* **2014**, *25*, 159. [\[CrossRef\]](#)
8. Peikert, C. An Efficient and Parallel Gaussian Sampler for Lattices. In Proceedings of the CRYPTO 2010, Santa Barbara, CA, USA, 15–19 August 2010; Rabin, T., Ed.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6223, pp. 80–97.
9. Pppelmann, T.; Ducas, L.; Gneysu, T. Enhanced Lattice-Based Signatures on Reconfigurable Hardware. In Proceedings of the CHES 2014, Busan, Korea, 23–26 September 2014; Batina, L., Robshaw, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8731, pp. 353–370.
10. Micciancio, D.; Walter, M. Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time. In Proceedings of the CRYPTO 2017, Santa Barbara, CA, USA, 20–24 August 2017; Katz, J., Shacham, H., Eds.; Proceedings, Part II; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10402, pp. 455–485.
11. Saarinen, M.J.O. Arithmetic coding and blinding countermeasures for lattice signatures. *J. Cryptogr. Eng.* **2018**, *8*, 71–84. [\[CrossRef\]](#)
12. Karmakar, A.; Roy, S.S.; Reparaz, O.; Vercauteren, F.; Verbauwhede, I. Constant-Time Discrete Gaussian Sampling. *IEEE Trans. Comput.* **2018**, *67*, 1561–1571. [\[CrossRef\]](#)
13. Karmakar, A.; Roy, S.S.; Vercauteren, F.; Verbauwhede, I. Pushing the speed limit of constant-time discrete Gaussian sampling. A case study on the Falcon signature scheme. In Proceedings of the 56th DAC 2019, Las Vegas, NV, USA, 2–6 June 2019; ACM: New York, NY, USA, 2019; p. 88. [\[CrossRef\]](#)
14. Karney, C.F. Sampling Exactly from the Normal Distribution. *ACM Trans. Math. Softw.* **2016**, *42*, 3:1–3:14. [\[CrossRef\]](#)
15. Zhao, R.K.; Steinfeld, R.; Sakzad, A. FACCT: Fast, Compact, and Constant-Time Discrete Gaussian Sampler over Integers. *IEEE Trans. Comput.* **2020**, *69*, 126–137. [\[CrossRef\]](#)
16. Barthe, G.; Belaid, S.; Espitau, T.; Fouque, P.; Rossi, M.; Tibouchi, M. GALACTICS: Gaussian Sampling for Lattice-Based Constant-Time Implementation of Cryptographic Signatures, Revisited. In Proceedings of the 2019 ACM SIGSAC CCS 2019, London, UK, 11–15 November 2019; Cavallaro, L., Kinder, J., Wang, X., Katz, J., Eds.; ACM: New York, NY, USA, 2019; pp. 2147–2164. [\[CrossRef\]](#)
17. Micciancio, D.; Peikert, C. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller. In Proceedings of the EUROCRYPT 2012, Cambridge, UK, 15–19 April 2012; Pointcheval, D., Johansson, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7237, pp. 700–718.
18. Genise, N.; Micciancio, D. Faster Gaussian Sampling for Trapdoor Lattices with Arbitrary Modulus. In Proceedings of the EUROCRYPT 2018, Tel Aviv, Israel, 29 April–3 May 2018; Nielsen, J.B., Rijmen, V., Eds.; Proceedings, Part I; Springer: Berlin/Heidelberg, Germany, 2018; Volume 10820, pp. 174–203. [\[CrossRef\]](#)
19. Roy, S.S.; Vercauteren, F.; Verbauwhede, I. High Precision Discrete Gaussian Sampling on FPGAs. In Proceedings of the SAC 2013, Burnaby, BC, Canada, 14–16 August 2013; Lange, T., Lauter, K.E., Lisonek, P., Eds.; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8282, pp. 383–401.
20. Howe, J.; Khalid, A.; Rafferty, C.; Regazzoni, F.; O’Neill, M. On Practical Discrete Gaussian Samplers for Lattice-Based Cryptography. *IEEE Trans. Comput.* **2018**, *67*, 322–334. [\[CrossRef\]](#)
21. Aguilar-Melchor, C.; Albrecht, M.R.; Ricosset, T. Sampling from Arbitrary Centered Discrete Gaussians for Lattice-Based Cryptography. In Proceedings of the ACNS 2017, Kanazawa, Japan, 10–12 July 2017; Gollmann, D., Miyaji, A., Kikuchi, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10355, pp. 3–19. [\[CrossRef\]](#)
22. Buchmann, J.A.; Cabarcas, D.; Gpfert, F.; Hlsing, A.; Weiden, P. Discrete Ziggurat: A Time-Memory Trade-Off for Sampling from a Gaussian Distribution over the Integers. In Proceedings of the SAC 2013, Burnaby, BC, Canada, 14–16 August 2013; Lange, T., Lauter, K.E., Lisonek, P., Eds.; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8282, pp. 402–417. [\[CrossRef\]](#)