Sampling From Arbitrary Centered Discrete Gaussians For Lattice-Based Cryptography

Carlos Aguilar-Melchor[1], Martin R. Albrecht[2⋆], and Thomas Ricosset[1,3]

[1] INP ENSEEIHT, IRIT-CNRS, Université de Toulouse, Toulouse, France
{carlos.aguilar,thomas.ricosset}@enseeiht.fr
[2] Information Security Group, Royal Holloway, University of London, London, United Kingdom
martin.albrecht@royalholloway.ac.uk
[3] Thales Communications & Security, Gennevilliers, France

Abstract. Non-Centered Discrete Gaussian sampling is a fundamental building block in many lattice-based constructions in cryptography, such as signature and identity-based encryption schemes. On the one hand, the center-dependent approaches, e.g. cumulative distribution tables (CDT), Knuth-Yao, the alias method, discrete Zigurat and their variants, are the fastest known algorithms to sample from a discrete Gaussian distribution. However, they use a relatively large precomputed table for each possible real center in $[0, 1)$ making them impracticable for non-centered discrete Gaussian sampling. On the other hand, rejection sampling allows to sample from a discrete Gaussian distribution for all real centers without prohibitive precomputation cost but needs costly floating-point arithmetic and several trials per sample. In this work, we study how to reduce the number of centers for which we have to precompute tables and propose a non-centered CDT algorithm with practicable size of precomputed tables as fast as its centered variant. Finally, we provide some experimental results for our open-source C++ implementation indicating that our sampler increases the rate of Peikert's algorithm for sampling from arbitrary lattices (and cosets) by a factor 3 with precomputation storage up to 6.2 MB.

1 Introduction

Lattice-based cryptography has generated considerable interest in the last decade due to many attractive features, including conjectured security against quantum attacks, strong security guarantees from worst-case hardness and constructions of fully homomorphic encryption (FHE) schemes (see the survey [33]). Moreover, lattice-based cryptographic schemes are often algorithmically simple and efficient, manipulating essentially vectors and matrices or polynomials modulo relatively small integers, and in some cases outperform traditional systems.

⋆ The research of this author was supported by EPSRC grant "Bit Security of Learning with Errors for Post-Quantum Cryptography and Fully Homomorphic Encryption" (EP/P009417/1) and the EPSRC grant "Multilinear Maps in Cryptography" (EP/L018543/1).

Modern lattice-based cryptosystems are built upon two main average-case problems over general lattices: Short Integer Solution (SIS) [1] and Learning With Errors (LWE) [36], and their analogues over ideal lattices, ring-SIS [29] and ring-LWE [27]. The hardness of these problems can be related to the one of their worst-case counterpart, if the instances follow specific distributions and parameters are choosen appropriately [1, 36, 29, 27].

In particular, discrete Gaussian distributions play a central role in lattice-based cryptography. A natural set of examples to illustrate the importance of Gaussian sampling are lattice-based signature and identity-based encryption (IBE) schemes [16]. The most iconic example is the signature algorithm proposed in [16] (hereafter GPV), as a secure alternative to the well-known (and broken) GGH signature scheme [18]. In this paper, the authors use the Klein/GPV algorithm [21], a randomized variant of Babai's nearest plane algorithm [4]. In this algorithm, the rounding step is replaced by randomized rounding according to a discrete Gaussian distribution to return a lattice point (almost) independent of a hidden basis. The GPV signature scheme has also been combined with LWE to obtain the first identity-based encryption (IBE) scheme [16] conjectured to be secure against quantum attacks. Later, a new Gaussian sampling algorithm for arbitrary lattices was presented in [32]. It is a randomized variant of Babai's rounding-off algorithm, is more efficient and parallelizable, but it outputs longer vectors than Klein/GPV's algorithm.

Alternatively to the above trapdoor technique, lattice-based signatures [26, 23, 24, 25, 11] were also constructed by applying the Fiat-Shamir heuristic [14]. Note that in contrast to the algorithms outlined above which sample from a discrete Gaussian distribution for any real center not known in advance, the schemes developed in [25, 11] only need to sample from a discrete Gaussian centered at zero.

1.1 Our Contributions

We develop techniques to speed-up discrete Gaussian sampling when the center is not known in advance, obtaining a flexible time-memory trade-off comparing favorably to rejection sampling. We start with the cumulative distribution table (CDT) suggested in [32] and lower the computational cost of the precomputation phase and the global memory required when sampling from a non-centered discrete Gaussian by precomputing the CDT for a relatively small number of centers, in $\mathcal{O}(\lambda^3)$, and by computing the cdf when needed, i.e. when for a given uniform random input, the values returned by the CDTs for the two closest precomputed centers differ. Second, we present an adaptation of the lazy technique described in [12] to compute most of the cdf in double IEEE standard double precision, thus decreasing the number of precomputed CDTs. Finally, we propose a more flexible approach which takes advantage of the information already present in the precomputed CDTs. For this we use a Taylor expansion around the precomputed centers and values instead of this lazy technique, thus enabling to reduce the number of precomputed CDTs to a $\omega(\lambda)$.

We stress, though, that our construction is not constant time, which limits its utility. We consider addressing this issue important future work.

1.2 Related Work

Many discrete Gaussian samplers over the Integers have been proposed for lattice-based cryptography. Rejection Sampling [17, 12], Inversion Sampling with a Cumulative Distribution Table (CDT) [32], Knuth-Yao [13], Discrete Ziggurat [7], Bernoulli Sampling [11], Kahn-Karney [20] and Binary Arithmetic Coding [37].

The optimal method will of course depend on the setting in which it is used. In this work, we focus on what can be done on a modern computer, with a comfortable amount of memery and hardwired integer and floating-point operations. This is in contrast to the works [11, 13] which focus on circuits or embedded devices. We consider exploring the limits of the usual memory and hardwired operations in commodity hardware as much an interesting question as it is to consider what is feasible in more constrained settings.

Rejection Sampling and variants. Straightforward rejection sampling [38] is a classical method to sample from any distribution by sampling from a uniform distribution and accept the value with a probability equal to its probability in the target distribution. This method does not use pre-computed data but needs floating-point arithmetic and several trials by sample. Bernoulli sampling [11] introduces an exponential bias from Bernouilli variables, which can be efficiently sampled specially in circuits. The bias is then corrected in a rejection phase based on another Bernouilli variable. This approach is particularly suited for embedded devices for the simplicity of the computation and the near-optimal entropy consumption. Kahn-Karney sampling is another variant of rejection sampling to sample from a discrete Gaussian distribution which does not use floating-point arithmetic. It is based on the von Neumann algorithm to sample from the exponential distribution [31], requires no precomputed tables and consumes a smaller amount of random bits than Bernoulli sampling, though it is slower. Currently the fastest approach in the computer setting uses a straightforward rejection sampling approach with "lazy" floating-point computations [12] using IEEE standard double precision floating-point numbers in most cases.

Note that none of these methods requires precomputation depending on the distribution's center c. In all the alternative approaches we present hereafter, there is some center-dependent precomputation. When the center is not know this can result in prohibitive costs and handling these becomes a major issue around which most of our work is focused.

Center-dependent approaches. The cumulative distribution table algorithm is based on the inversion method [9]. All non-negligible cumulative probabilities are stored in a table and at sampling time one generates a cumulative probability in $[0, 1)$ uniformly at random, performs a binary search through the table and returns the corresponding value. Several alternatives to straightforward CDT

are possible. Of special interest are: the alias method [39] which encodes CDTs in a more involved but more efficient approach; BAC Sampling [37] which uses arithmetic coding tables to sample with an optimal consumption of random bits; and Discrete Ziggurat [7] which adapts the Ziggurat method [28] for a flexible time-memory trade-off. Knuth-Yao sampling [22] uses a random bit generator to traverse a binary tree formed from the bit representation of the probability of each possible sample, the terminal node is labeled by the corresponding sample. The main advantage of this method is that it consumes a near-optimal amount of random bits. A block variant and other practical improvements are suggested in [13]. This method is center-dependent but clearly designed for circuits and on a computer setting it is surpassed by other approaches.

Our main contribution is to show how to get rid of the known-center constraint with reasonable memory usage for center-dependent approaches. As a consequence, we obtain a performance gain with respect to rejection sampling approaches. Alternatively, any of the methods discussed above could have replaced our straightforward CDT approach. This, however, would have made our algorithms, proofs, and implementations more involved. On the other hand, further performance improvements could perhaps be achieved this way. This is an interesting problem for future work.

2 Preliminaries

Throughout this work, we denote the set of real numbers by $\mathbb{R}$ and the Integers by $\mathbb{Z}$. We extend any real function $f(\cdot)$ to a countable set A by defining $f(A) = \sum_{x \in A} f(x)$. We denote also by U_I the uniform distribution on I.

2.1 Discrete Gaussian Distributions on $\mathbb{Z}$

The discrete Gaussian distribution on $\mathbb{Z}$ is defined as the probability distribution whose unnormalized density function is

$$\begin{aligned} \rho : \mathbb{Z} &\to [0, 1) \\ x &\to e^{\frac{-x^2}{2}} \end{aligned}$$

If $s \in \mathbb{R}^+$ and $c \in \mathbb{R}$, then we extend this definition to

$$\rho_{s,c}(x) := \rho\left(\frac{x-c}{s}\right)$$

and denote $\rho_{s,0}(x)$ by $\rho_s(x)$. For any mean $c \in \mathbb{R}$ and parameter $s \in \mathbb{R}^+$ we can now define the discrete Gaussian distribution $D_{s,c}$ as

$$\forall x \in \mathbb{Z},\ D_{s,c}(x) := \frac{\rho_{s,c}(x)}{\rho_{s,c}(\mathbb{Z})}$$

Note that the standard deviation of this distribution is $\sigma = s/\sqrt{2\pi}$. We also define $\mathrm{cdf}_{s,c}$ as the cumulative distribution function (cdf) of $D_{s,c}$

$$\forall x \in \mathbb{Z},\ \mathrm{cdf}_{s,c}(x) := \sum_{i=-\infty}^{x} D_{s,c}(i)$$

Smoothing Parameter. The smoothing parameter $\eta_\epsilon(\Lambda)$ quantifies the minimal discrete Gaussian parameter s required to obtain a given level of smoothness on the lattice Λ. Intuitively, if one picks a noise vector over a lattice from a discrete Gaussian distribution with radius at least as large as the smoothing parameter, and reduces this modulo the fundamental parallelepiped of the lattice, then the resulting distribution is very close to uniform (for details and formal definition see [30]).

Gaussian Measure. An interesting property of discrete Gaussian distributions with a parameter s greater than the smoothing parameter is that the Gaussian measure, i.e. $\rho_{s,c}(\mathbb{Z})$ for $D_{s,c}$, is essentially the same for all centers.

Lemma 1 (From the proof of [30, Lemma 4.4]). *For any $\epsilon \in (0,1)$, $s > \eta_\epsilon(\mathbb{Z})$ and $c \in \mathbb{R}$ we have*

$$\Delta_{\mathsf{measure}} := \frac{\rho_{s,c}(\mathbb{Z})}{\rho_{s,0}(\mathbb{Z})} \leq \frac{1-\epsilon}{1+\epsilon}$$

Tailcut Parameter. To deal with the infinite domain of Gaussian distributions, algorithms usually take advantage of their rapid decay to sample from a finite domain. The next lemma is useful in determining the tailcut parameter τ.

Lemma 2 ([17, Lemma 4.2]). *For any $\epsilon > 0$, $s > \eta_\epsilon(\mathbb{Z})$ and $\tau > 0$, we have*

$$\mathsf{E}_{\mathsf{tailcut}} := \Pr_{X \sim D_{\mathbb{Z},s,c}}[|X - c| > \tau s] < 2e^{-\pi\tau^2} \cdot \frac{1+\epsilon}{1-\epsilon}$$

2.2 Floating-Point Arithmetic

We recall some facts from [12] about floating-point arithmetic (FPA) with m bits of mantissa, which we denote by $\mathbb{FP}_m$. A floating-point number is a triplet $\bar{x} = (s, e, v)$ where $s \in \{0,1\}$, $e \in \mathbb{Z}$ and $v \in \mathbb{N}_{2^m-1}$ which represents the real number $\bar{x} = (-1)^s \cdot 2^{e-m} \cdot v$. Denote by $\epsilon = 2^{1-m}$ the floating-point precision. Every FPA-operation $\bar{\circ} \in \{\bar{+}, \bar{-}, \bar{\times}, \bar{/}\}$ and its respective arithmetic operation on $\mathbb{R}$, $\circ \in \{+, -, \times, /\}$ verify

$$\forall \bar{x}, \bar{y} \in \mathbb{FP}_m, |(\bar{x} \ \bar{\circ}\ \bar{y}) - (\bar{x} \circ \bar{y})| \leq (x \circ y)\epsilon$$

Moreover, we assume that the floating-point implementation of the exponential function $\bar{\exp}(\cdot)$ verifies

$$\forall \bar{x} \in \mathbb{FP}_m, |\bar{\exp}(\bar{x}) - \exp(\bar{x})| \leq \epsilon$$

2.3 Taylor Expansion

Taylor's theorem provides a polynomial approximation around a given point for any function sufficiently differentiable.

Theorem 1 (Taylor's theorem). *Let $d \in \mathbb{Z}^+$ and let the function $f : \mathbb{R} \to \mathbb{R}$ be d times differentiable in some neighborhood U of $a \in \mathbb{R}$. Then for any $x \in U$*

$$f(x) = \mathcal{T}_{d,f,a}(x) + \mathcal{R}_{d,f,a}(x)$$

where

$$\mathcal{T}_{d,f,a}(x) = \sum_{i=0}^{d} \frac{f^{(i)}(a)}{i!}(x-a)^i$$

and

$$\mathcal{R}_{d,f,a}(x) = \int_a^x \frac{f^{(d+1)}(t)}{d!}(x-t)^d dt$$

3 Variable-Center with Polynomial Number of CDTs

We consider the case in which the mean is variable, i.e. the center is not know before the online phase, as it is the case for lattice-based hash-and-sign signatures. The center can be any real number, but without loss of generality we will only consider centers in $[0, 1)$. Because CDTs are center-dependent, a first naive option would be to precompute a CDT for each possible real center in $[0, 1)$ in accordance with the desired accuracy. Obviously, this first option has the same time complexity than the classical CDT algorithm, i.e. $\mathcal{O}(\lambda \log s\lambda)$ for λ the security parameter. However, it is completely impractical with 2^λ precomputed CDTs of size $\mathcal{O}(s\lambda^{1.5})$. An opposite trade-off is to compute the CDT on-the-fly, avoiding any precomputation storage, which increase the computational cost to $\mathcal{O}(s\lambda^{3.5})$ assuming that the computation of the exponential function run in $\mathcal{O}(\lambda^3)$ (see Section 3.2 for a justification of this assumption).

An interesting question is can we keep the time complexity of the classical CDT algorithm with a polynomial number of precomputed CDTs. To answer this question, we start by fixing the number n of equally spaced centers in $[0, 1)$ and precompute the CDTs for each of these. Then, we apply the CDT algorithm to the two precomputed centers closest to the desired center for the same cumulative probability uniformly draw. Assuming that the number of precomputed CDTs is sufficient, the values returned from both CDTs will be equal most of the time, in this case we can conclude, thanks to a simple monotonic argument, that the returned value would have been the same for the CDT at the desired center and return it as a valid sample. Otherwise, the largest value will immediately follow the smallest and we will then have to compute the cdf at the smallest value for the desired center in order to know if the cumulative probability is lower or higher than this cdf. If it is lower then the smaller value will be returned as sample, else it will be the largest.

3.1 Twin-CDT Algorithm

As discussed above, to decrease the memory required by the CDT algorithm when the distribution center is determined during the online phase, we can precompute CDTs for a number n of centers equally spaced in $[0, 1)$ and compute the cdf when necessary. Algorithm 1 resp. 2 describes the offline resp. online phase of the *Twin-CDT* algorithm. Algorithm 1 precomputes CDTs, up to a

Algorithm 1 Twin-CDT Algorithm: Offline Phase

Input: a Gaussian parameter s and a number of centers n
Output: a precomputed matrix $\mathbf{T}$

initialize an empty matrix $\mathbf{T} \in \mathbb{FP}_\lambda^{n \times 2\lceil \tau s \rceil + 3}$
for $i \leftarrow 0, \ldots, n-1$ **do**
    for $j \leftarrow 0, \ldots, 2\lceil \tau s \rceil + 2$ **do**
        $\mathbf{T}_{i,j} \leftarrow \mathbb{FP}_m : \mathrm{cdf}_{s,i/n}(j - \lceil \tau s \rceil - 1)$

precision m that guarantees the λ most significant bits of each cdf, and store them with λ-bits of precision as a matrix $\mathbf{T}$, where the i-th line is the CDT corresponding to the i-th precomputed center i/n. To sample from $D_{s,c}$, Algo-

Algorithm 2 Twin-CDT Algorithm: Online Phase

Input: a center c and a precomputed matrix $\mathbf{T}$
Output: a sample x that follows $D_{s,c}$

$p \leftarrow U_{[0,1) \cap \mathbb{FP}_\lambda}$
$v_1 \leftarrow i - \lceil \tau s \rceil - 1$ s.t. $\mathbf{T}_{\lfloor n(c - \lfloor c \rfloor) \rfloor, i-1} \leq p < \mathbf{T}_{\lfloor n(c - \lfloor c \rfloor) \rfloor, i}$
$v_2 \leftarrow j - \lceil \tau s \rceil - 1$ s.t. $\mathbf{T}_{\lceil n(c - \lfloor c \rfloor) \rceil, j-1} \leq p < \mathbf{T}_{\lceil n(c - \lfloor c \rfloor) \rceil, j}$
if $v_1 = v_2$ **then**
    return $v_1 + \lfloor c \rfloor$
else
    if $p < \mathbb{FP}_m : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
        return $v_1 + \lfloor c \rfloor$
    else
        return $v_2 + \lfloor c \rfloor$

rithm 2 searches the preimages by the cdf of a cumulative probability p, draw from the uniform distribution on $[0, 1) \cap \mathbb{FP}_\lambda$, in both CDTs corresponding to the center $\lfloor n(c - \lfloor c \rfloor) \rfloor / n$ (respectively $\lceil n(c - \lfloor c \rfloor) \rceil / n$) which return a value v_1 (resp. v_2). If the same value is returned from the both CDTs (i.e. $v_1 = v_2$), then this value added the desired center integer part is a valid sample, else it computes $\mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ and returns $v_1 + \lfloor c \rfloor$ if $p < \mathrm{cdf}_{s,c}(v_1)$ and $v_2 + \lfloor c \rfloor$ else.

Correctness. We establish correctness in the lemma below.

Lemma 3. *Assuming that m is large enough to ensure λ correct bits during the cdf computation, the statistical distance between the output distribution of Algorithm 2 instantiated to sample from $D_{\mathbb{Z}^m,\sigma,c}$ and $D_{\mathbb{Z}^m,\sigma,c}$ is bounded by $2^{-\lambda}$.*

Proof. First note that from the discrete nature of the considered distribution we have $D_{s,c} = D_{s,c-\lfloor c \rfloor} + \lfloor c \rfloor$. Now recall that the probability integral transform states that if X is a continuous random variable with cumulative distribution function cdf, then $\mathrm{cdf}(X)$ has a uniform distribution on $[0,1]$. Hence the inversion method: $\mathrm{cdf}^{-1}(U_{[0,1]})$ has the same distribution as X. Finally by noting that for all $s, p \in \mathbb{R}$, $\mathrm{cdf}_{s,c}(p)$ is monotonic in c, if $\mathrm{cdf}^{-1}_{s,c_1}(p) = \mathrm{cdf}^{-1}_{s,c_2}(p) := v$, then $\mathrm{cdf}^{-1}_{s,c}(p) = v$ for all $c \in [c_1, c_2]$, and as a consequence, for all $v \in [-\lceil \tau s \rceil - 1, \lceil \tau s \rceil + 1]$, the probability of outputting v is equal to $\mathbb{FP}_m : \mathrm{cdf}_{s,c}(v) - \mathbb{FP}_m : \mathrm{cdf}_{s,c}(v-1)$ which is $2^{-\lambda}$-close to $D_{s,c}(v)$. □

The remaining issue in the correctness analysis of Algorithm 2 is to determine the error occurring during the m-precision cdf computation. Indeed, this error allows us to learn what precision m is needed to correctly compute the λ most significant bits of the cdf. This error is characterized in Lemma 4.

Lemma 4. *Let $m \in \mathbb{Z}$ be a positive integer and $\varepsilon = 2^{1-m}$. Let $\bar{c}, \bar{s}, \bar{h} \in \mathbb{FP}_m$ be at distance respectively at most δ_c, δ_c and δ_h from $c, s, h \in \mathbb{R}$ and $h = 1/\rho_{s,c}(\mathbb{Z})$. Let $\Delta f(x) := |\mathbb{FP}_m : f(x) - f(x)|$. We also assume that the following inequalities hold: $s \geq 4$, $\tau \geq 10$, $s\delta_s \leq 0.01$, $\delta_c \leq 0.01$, $s^2\varepsilon \leq 0.01$, $(\tau s + 1)\varepsilon \leq 1/2$. We have the following error bound on $\Delta\mathrm{cdf}_{s,c}(x)$ for any integer x such that $|x| \leq \tau s + 2$*

$$\Delta\mathrm{cdf}_{s,c}(x) \leq 3.5\tau^3 s^2 \varepsilon$$

Proof. We derive the following bounds using [10, Facts 6.12, 6.14, 6.22]:

$$\Delta\mathrm{cdf}_{s,c}(x) \leq \Delta\left[\sum_{i=-\lceil \tau s \rceil - 1}^{\lceil \tau s \rceil + 1} \rho_{s,c}(i)\right]\left(\frac{1}{s} + 3.6s\varepsilon\right) + 3.6s\varepsilon$$

$$\Delta\left[\sum_{i=-\lceil \tau s \rceil - 1}^{\lceil \tau s \rceil + 1} \rho_{s,c}(i)\right] \leq 3.2\tau^3 s^3 \varepsilon$$

□

For the sake of readability the FPA error bound of Lemma 4 is fully simplified and is therefore not tight. For practical implementation, one can derive a better bound using an ad-hoc approach such as done in [35].

Efficiency. On average, the evaluation of the cdf requires $\lceil \tau s \rceil + 1.5$ evaluations of the exponential function. For the sake of clarity, we assume that the exponential function is computed using a direct power series evaluation with schoolbook multiplication, so its time complexity is $\mathcal{O}(\lambda^3)$. We refer the reader to [6] for a discussion of different ways to compute the exponential function in high-precision.

Lemma 5 establishes that the time complexity of Algorithm 2 is $\mathcal{O}(\lambda \log s\lambda + \lambda^4/n)$, so with $n = \mathcal{O}(\lambda^3)$ it has asymptotically the same computational cost than the classical CDT algorithm.

Lemma 5. *Let P_{cdf} be the probability of computing the* cdf *during the execution of Algorithm 2, assuming that $\tau s \geq 10$, we have*

$$\mathsf{P}_{cdf} \leq 2.2\tau s \left(1 - e^{-\frac{1.25\tau}{sn}} \Delta_{measure}\right)$$

Proof.

$$\mathsf{P}_{\mathsf{cdf}} \leq \max_{c \in [0,1)} \left(\sum_{i=-\lceil \tau s \rceil - 1}^{\lceil \tau s \rceil + 1} \left| \mathrm{cdf}_{s,c}(i) - cdf_{s,c+\frac{1}{n}}(i) \right| \right)$$

Assuming that $\tau s \geq 10$, we have

$$e^{-\frac{1.25\tau}{sn}} \Delta_{\mathsf{measure}} \mathrm{cdf}_{s,c}(i) \leq \mathrm{cdf}_{s,c+\frac{1}{n}}(i) \leq \mathrm{cdf}_{s,c}(i)$$

Hence the upper bound. □

On the other hand, the precomputation matrix generated by Algorithm 1 take n times the size of one CDT, hence the space complexity is $\mathcal{O}(ns\lambda^{1.5})$. Note that for n sufficiently big to make the cdf computational cost negligible, the memory space required by this algorithm is about 1 gigabyte for the parameters considered in cryptography and thus prohibitively expensive for practical use.

3.2 Lazy-CDT Algorithm

A first idea to decrease the number of precomputed CDTs is to avoid costly cdf evaluations by using the same lazy trick as in [12] for rejection sampling. Indeed, a careful analysis of Algorithm 2 shows most of the time many of the computed cdf bits are not used. This gives us to a new strategy which consists of computing the bits of $\mathrm{cdf}_{s,c}(v_1)$ lazily. When the values corresponding to the generated probability for the two closest centers are different, the *Lazy-CDT* algorithm first only computes the cdf at a precision m' to ensure $k < \lambda$ correct bits. If the comparison is decided with those k bits, it returns the sample. Otherwise, it recomputes the cdf at a precision m to ensure λ correct bits.

Correctness. In addition to the choice of m, discussed in Section 3.1, to achieve λ bits of precision, the correctness of Algorithm 3 also requires to know k which is the number of correct bits after the floating-point computation of the cdf with m' bits of mantissa. For this purpose, given m' Lemma 4 provides a theoretical lower bound on k.

Algorithm 3 Lazy-CDT Algorithm: Online Phase

Input: a center c and a precomputed matrix $\mathbf{T}$
Output: a sample x that follows $D_{s,c}$

$p \leftarrow U_{[0,1)\cap\mathbb{FP}_\lambda}$
$v_1 \leftarrow i - \lceil \tau s \rceil - 1$ s.t. $\mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i-1} \leq p < \mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i}$
$v_2 \leftarrow j - \lceil \tau s \rceil - 1$ s.t. $\mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j-1} \leq p < \mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j}$
if $v_1 = v_2$ **then**
    return $v_1 + \lfloor c \rfloor$
else
    if $\mathbb{FP}_k : p < \mathbb{FP}_{m'} : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
        return $v_1 + \lfloor c \rfloor$
    else
        if $\mathbb{FP}_k : p > \mathbb{FP}_{m'} : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
            return $v_2 + \lfloor c \rfloor$
        else
            if $p > \mathbb{FP}_m : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
                return $v_1 + \lfloor c \rfloor$
            else
                return $v_2 + \lfloor c \rfloor$

Efficiency. As explained in [12] the precision used for floating-point arithmetic has non-negligible impact, because fp-operation become much expensive when the precision goes over the hardware precision. For instance, modern processors typically provide floating-point arithmetic following the double IEEE standard double precision ($m = 53$), but quad-float FPA ($m = 113$) is usually about 10-20 times slower for basic operations, and the overhead is much more for multiprecision FPA. Therefore the maximal hardware precision is a natural choice for m'. However this choice for m' in Algorithm 3 is a strong constraint for cryptographic applications, where the error occurring during the floating-point cdf computation is usually greater than 10 bits, making the time-memory tradeoff of Algorithm 3 inflexible. Note that the probability of triggering high precision in Algorithm 3 given that $v_1 \neq v_2$ is about $2^{q-k}\mathsf{P}_{\mathsf{cdf}}$, where q is the number of common leading bits of $\mathrm{cdf}_{s,\lfloor n(c-\lfloor c \rfloor)\rfloor/n}(v_1)$ and $\mathrm{cdf}_{s,\lceil n(c-\lfloor c \rfloor)\rceil/n}(v_2)$. By using this lazy trick in addition to lookup tables as described in Section 5 with parameters considered in cryptography, we achieve a computational cost lower than the classical centered CDT algorithm with a memory requirement in the order of 1 megabyte.

4 A More Flexible Time-Memory Tradeoff

In view of limitations of the lazy approach described above, a natural question is if we can find a better solution to approximate the cdf. The major advantage of this lazy trick is that it does not require additional memory. However, in our context the CDTs are precomputed and rather than approximate the cdf from scratch it would be interesting to reuse the information contained in these

precomputations. Consider the cdf as a function of the center and note that each precomputed cdf is zero degree term of the Taylor expansion of the cdf around a precomputed center. Hence, we may approximate the cdf by its Taylor expansions by precomputing some higher degree terms.

At a first glance, this seems to increase the memory requirements of the sampling algorithm, but we will show that this approach allows to drastically reduce the number of precomputed to a $\omega(\lambda)$ centers thanks to a probability which decreases rapidly with the degree of the Taylor expansion. Moreover, this approximation is faster than the cdf lazy computation and it has no strong constraints related to the maximal hardware precision. As a result, we obtain a flexible time-memory tradeoff which reaches, in particular, the same time complexity as the CDT algorithm for centered discrete Gaussians with a practical memory requirements for cryptographic parameters.

4.1 Taylor-CDT Algorithm

Our *Taylor-CDT* algorithm is similar to the *Lazy-CDT* algorithm (Algorithm 3) described above, except that the lazy computation of the cdf is replaced by the Taylor expansion of the cdf, viewed as a function of the Gaussian center, around each precomputed centers for all possible values. The zero-degree term of each of these Taylor expansions is present in the corresponding CDT element $\mathbf{T}_{i,j}$ and the d higher-degree terms are stored as an element $\mathbf{E}_{i,j}$ of another matrix $\mathbf{E}$. As for the other approaches, these precomputations shall be performed at a sufficient precision m to ensure λ correct bits. During the online phase, Algorithm 5 proceed as follow. Draw p from the uniform distribution over $[0,1) \cap \mathbb{FP}_\lambda$ and search p in the CDTs of the two closest precomputed centers to the desired center decimal part. If the two values found are equal, add the desired center integer part to this value and return it as a valid sample. Otherwise, select the closest precomputed center to the desired center decimal part and evaluate, at the desired center decimal part, the Taylor expansion corresponding to this center and the value found in its CDT. If p is smaller or bigger than this evaluation with respect for the error approximation upper bound $\mathsf{E}_{\mathsf{expansion}}$, characterized in Lemma 6, add the desired center integer part to the corresponding value and

Algorithm 4 Taylor-CDT Algorithm: Offline Phase

Input: a Gaussian parameter s, a number of centers n, a Taylor expansion degree d
Output: two precomputed matrices $\mathbf{T}$ and $\mathbf{E}$

1: initialize two empty matrices $\mathbf{T} \in \mathbb{FP}_\lambda^{n \times 2\lceil \tau s \rceil + 3}$ and $\mathbf{E} \in (\mathbb{FP}_\lambda^d)^{n \times 2\lceil \tau s \rceil + 3}$
2: **for** $i \leftarrow 0, \ldots, n-1$ **do**
3: **for** $j \leftarrow 0, \ldots, 2\lceil \tau s \rceil + 2$ **do**
4: $\mathbf{T}_{i,j} \leftarrow \mathbb{FP}_m : \mathrm{cdf}_{s,i/n}(j - \lceil \tau s \rceil - 1)$
5: $\mathbf{E}_{i,j} \leftarrow \mathbb{FP}_m : \mathcal{T}_{d,\mathrm{cdf}_{s,x}(j-\lceil \tau s \rceil - 1),i/n}(x) - \mathbf{T}_{i,j}$

return it as a valid sample. Otherwise, it is necessary to compute the full cdf to decide which value to return.

Algorithm 5 Taylor-CDT Algorithm: Online Phase

Input: a center c and two precomputed matrices $\mathbf{T}$ and $\mathbf{E}$
Output: a sample x that follows $D_{s,c}$

$p \leftarrow U_{[0,1)\cap\mathbb{FP}_\lambda}$
$v_1 \leftarrow i - \lceil \tau s \rceil - 1$ s.t. $\mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i-1} \leq p < \mathbf{T}_{\lfloor n(c-\lfloor c \rfloor)\rfloor, i}$
$v_2 \leftarrow j - \lceil \tau s \rceil - 1$ s.t. $\mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j-1} \leq p < \mathbf{T}_{\lceil n(c-\lfloor c \rfloor)\rceil, j}$
if $v_1 = v_2$ **then**
    return $v_1 + \lfloor c \rfloor$
else
    if $|c - \lfloor n(c - \lfloor c \rfloor)\rfloor| < |c - \lceil n(c - \lfloor c \rfloor)\rceil|$ **then**
        $c' \leftarrow \lfloor n(c - \lfloor c \rfloor)\rfloor$
    else
        $c' \leftarrow \lceil n(c - \lfloor c \rfloor)\rceil$
        $i \leftarrow j$
    if $p < \mathbf{T}_{c',i} + \mathbf{E}_{c',i}(c - \lfloor c \rfloor) - \mathsf{E}_{\mathsf{expansion}}$ **then**
        return $v_1 + \lfloor c \rfloor$
    else
        if $p > \mathbf{T}_{c',i} + \mathbf{E}_{c',i}(c - \lfloor c \rfloor) + \mathsf{E}_{\mathsf{expansion}}$ **then**
            return $v_2 + \lfloor c \rfloor$
        else
            if $p > \mathbb{FP}_m : \mathrm{cdf}_{s,c-\lfloor c \rfloor}(v_1)$ **then**
                return $v_1 + \lfloor c \rfloor$
            else
                return $v_2 + \lfloor c \rfloor$

Efficiency. Algorithm 5 performs two binary searches on CDTs in $\mathcal{O}(\lambda \log s\lambda)$, d additions and multiplications on $\mathbb{FP}_m$ in $\mathcal{O}(m^2)$ with probability $\mathsf{P}_{\mathsf{cdf}} \approx 3\lambda/n$ (see Lemma 5) and a cdf computation on $\mathbb{FP}_m$ in $\mathcal{O}(s\lambda^{3.5})$ with probability close to $2^{q+1}\mathsf{P}_{\mathsf{cdf}}\mathsf{E}_{\mathsf{expansion}}$, where q is the number of common leading bits of $\mathrm{cdf}_{s,\lfloor n(c-\lfloor c \rfloor)\rfloor/n}(v_1)$ and $\mathrm{cdf}_{s,\lceil n(c-\lfloor c \rfloor)\rceil/n}(v_2)$ and $\mathsf{E}_{\mathsf{expansion}}$ is the Taylor expansion approximation error bound described in Lemma 6.

Lemma 6. *Let $\mathsf{E}_{\mathsf{expansion}}$ be the maximal Euclidean distance between $\mathrm{cdf}_{s,x}(v)$ and $\mathcal{T}_{d,\mathrm{cdf}_{s,x}(v),c}(x)$, its Taylor expansion around c, for all $v \in [-\lceil \tau s \rceil - 1, \lceil \tau s \rceil + 1]$, $c \in [0,1)$ and $x \in [c, c + 1/2n]$, assuming that $\tau \geq 2.5$, $s \geq 4$, we have*

$$\mathsf{E}_{\mathsf{expansion}} < \frac{4\tau^{d+2}}{n^{d+1} s^{\frac{d+1}{2}}}$$

Proof. From Theorem 1 we have

$$\mathsf{E}_{\mathsf{expansion}} = \max_{\substack{c\in[0,1)\\ x\in[c,c+1/2n]\\ v\in[-\lceil\tau s\rceil-1,\lceil\tau s\rceil+1]}} \left(\sum_{i=-\lceil\tau s\rceil-1}^{v} \int_c^x \frac{\rho_{s,t}^{(d+1)}(i)}{d!\,\rho_{s,t}(\mathbb{Z})} (c + \frac{1}{2n} - t)^d \, dt \right)$$

By using well-known series-integral comparison we obtain $\rho_{s,t}(\mathbb{Z}) \geq s\sqrt{2\pi} - 1$ and since $\left|\rho_{s,t}^{(d)}(i)\right| < \frac{d(1.3\tau)^d 2^d}{s^{d/2}}$ for $s \geq 4$ and $\tau \geq 2.5$, it follows that

$$\mathsf{E}_{\mathsf{expansion}} \leq \frac{(d+1)(1.3)^{d+1}\tau^{d+2}}{d!\,n^{d+1}s^{\frac{d+1}{2}}}$$

□

A careful analysis of this technique show that with $d = 4$ we achieve the same asymptotic computational cost as the classical CDT algorithm with $n = \omega(\lambda)$, where the hidden factor is less than $1/4$, therefore for this degree the space complexity of Algorithm 4 and 5 is only λ times bigger than for centered sampling, showing that these algorithms can achieve a memory requirement as low as 1 MB. Finally, note that taking care to add the floating-point computation error to the error of approximation, one can compute the Taylor expansion evaluation at the maximal hardware precision to reduce its computational cost.

5 Lookup Tables

We shall now show how to use partial lookup tables to avoid the binary search in most cases when using CDT algorithms, this technique is the CDT analogue of the Knuth-Yao algorithm improvement described in [8]. Note that this strategy is particularly fitting for discrete Gaussian distributions with relatively small expected values. The basic idea is to subdivide the uniform distribution $U_{[0,1)}$ into ℓ uniform distributions on subsets of the same size $U_{[i/\ell,(i+1)/\ell)}$, with ℓ a power of two. We then precompute a partial lookup table on these subsets which allows to return the sample at once when the subset considered does not include a cdf image. We note that instead of subdividing the uniform range into stripes of the same size, we can also recursively subdivide only some stripes of the previous subdivision. However, for the sake of clarity and ease of exposure, this improvement is not included in this paper and we will describe this technique for the classical centered CDT algorithm.

First, we initialize a lookup table of size $\ell = 2^l$ where the i-th entry corresponds to a subinterval $[i/\ell, (i+1)/\ell)$ of $[0, 1)$. Second, after precomputing the CDT, we mark all the entries for which there is at least one CDT element in their corresponding subinterval $[i/\ell, (i+1)/\ell)$ with $\perp$, and all remaining entries with $\top$. Each entry marked with $\top$ allows to return a sample without the need to perform a binary search in the CDT, because only one value corresponds to this subinterval which is the first CDT element greater or equal to $(i+1)/\ell$.

Efficiency. The efficiency of this technique is directly related to the number of entries, marked with $\top$, whose subintervals do not contain a CDT element. We denote the probability of performing binary search by $\mathsf{P}_{\mathsf{binsrch}}$, obviously the probability to return the sample immediately after choosing i, which is a part of p, is $1 - \mathsf{P}_{\mathsf{binsrch}}$. Lemma 7 gives a lower bound of $\mathsf{P}_{\mathsf{binsrch}}$.

Lemma 7. *For any $\ell \geq 2^8$ and $s \geq \eta_{\frac{1}{2}}(\mathbb{Z})$. Let $P_{binsrch}$ be the probability of performing binary search during the execution of the CDT algorithm implemented with the lookup table trick described above, we have*

$$P_{binsrch} < 1.2s\sqrt{\log_2 \ell}/\ell$$

Proof.

$$\mathsf{P}_{\mathsf{binsrch}} = \frac{\ell - \sum_{i=\lfloor c-\tau s \rfloor}^{\lceil c+\tau s \rceil} \lfloor \ell \operatorname{cdf}_{s,c}(i) \rfloor - \lfloor \ell \operatorname{cdf}_{s,c}(i-1) \rfloor}{\ell}$$

From Lemma 2 we have

$$\left\lfloor \ell \operatorname{cdf}_{s,c}\left(\left\lfloor c - 0.6s\sqrt{\log_2 \ell} \right\rfloor\right) \right\rfloor = 0$$

$$\left\lfloor \ell \left(1 - \operatorname{cdf}_{s,c}\left(\left\lceil c + 0.6s\sqrt{\log_2 \ell} \right\rceil\right)\right) \right\rfloor = 0$$

□

6 Experimental Results

In this section, we present experimental results of our C++ implementation[4] distributed under the terms of the GNU General Public License version 3 or later (GPLv3+) which uses the MPFR [15] and GMP [19] libraries as well as Salsa20 [5] as the pseudorandom number generator. Our non-centered discrete Gaussian sampler was implemented with a binary search executed byte by byte if $\ell = 2^8$ and 2-bytes by 2-bytes if $\ell = 2^{16}$ without recursive subdivision of $U_{[0,1)}$, therefore $[0, 1)$ is subdivided in ℓ intervals of the same size and $\operatorname{cdf}(x)$ is stored for all $x \in [-\lceil \tau\sigma \rceil - 1, \lceil \tau\sigma \rceil + 1]$. The implementation of our non-centered discrete Gaussian sampler uses a fixed number of precomputed centers $n = 2^8$ with a lookup table of size $\ell = 2^8$ and includes the lazy cdf evaluation optimization.

We tested the performance of our non-centered discrete Gaussian sampler by using it as a subroutine for Peikert's sampler [32] for sampling from $D_{(g),\sigma',0}$ with $g \in \mathbb{Z}[x]/(x^N + 1)$ for N a power of two. To this end, we adapted the implementation of this sampler from [3] where we swap out the sampler from the dgs library [2] (implementing rejection sampling and [11]) used in [3] with our sampler for sampling for $D_{\mathbb{Z},\sigma,c}$. Note that sampling from $D_{(g),\sigma',0}$ is more involved and thus slower than sampling from $D_{\mathbb{Z}^N,\sigma',0}$. That is, to sample from $D_{(g),\sigma',0}$, [3] first computes an approximate square root of $\Sigma_2 = \sigma'^2 \cdot g^{-T} \cdot g^{-1} - r^2$ with $r = 2 \cdot \lceil \sqrt{\log N} \rceil$. Then, given an approximation $\sqrt{\Sigma_2}'$ of $\sqrt{\Sigma_2}$ it samples

[4] The implementation is available at `https://github.com/tricosset/FGN`.

a vector $x \leftarrow_{\$} \mathbb{R}^N$ from a standard normal distribution and interpret it as a polynomial in $\mathbb{Q}[X]/(x^N+1)$; computes $y = \sqrt{\Sigma_2}' \cdot x$ in $\mathbb{Q}[X]/(x^N+1)$ and returns $g \cdot (\lfloor y \rceil_r)$, where $\lfloor y \rceil_r$ denotes sampling a vector in $\mathbb{Z}^N$ where the i-th component follows $D_{\mathbb{Z},r,y_i}$. Thus, implementing Peikert's sampler requires sampling from $D_{\mathbb{Z},r,y_i}$ for changing centers y_i and sampling from a standard normal distribution. We give experimental results in Table 1, indicating that our sampler increases the rate by a factor ≈ 3.

			[3]		
N	$\log \sigma'$	precomp	time	$D_{(g),\sigma'}/s$	memory
256	38.2	0.08 s	8.46 ms	118.17	11,556 kB
512	42.0	0.17 s	16.96 ms	58.95	11,340 kB
1024	45.8	0.32 s	38.05 ms	26.28	21,424 kB
2048	49.6	0.93 s	78.17 ms	12.79	41,960 kB
4096	53.3	2.26 s	157.53 ms	6.35	86,640 kB
8192	57.0	6.08 s	337.32 ms	2.96	192,520 kB
16384	60.7	13.36 s	700.75 ms	1.43	301,200 kB
			this work		
N	$\log \sigma'$	precomp	time	$D_{(g),\sigma'}/s$	memory
256	38.2	0.31 s	2.91 ms	343.16	17,080 kB
512	42.0	0.39 s	5.99 ms	166.88	21,276 kB
1024	45.8	0.65 s	11.89 ms	84.12	38,280 kB
2048	49.6	1.04 s	25.07 ms	39.89	74,668 kB
4096	53.3	2.35 s	48.63 ms	20.56	148,936 kB
8192	57.0	7.27 s	96.67 ms	10.34	302,616 kB
16384	60.7	14.41 s	205.35 ms	4.87	618,448 kB

Table 1. Performance of sampling from $D_{(g),\sigma'}$ as implemented in [3] and with our non-centered discrete Gaussian sampler with $\ell = n = 2^8$. The column $D_{(g),\sigma'}/s$ gives the number of samples returned per second, the column "memory" the maximum amount of memory consumed by the process. All timings are on a Intel(R) Xeon(R) CPU E5-2667 (strombenzin). Precomputation uses 2 cores, the online phase uses one core.

References

1. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing. pp. 99–108. STOC '96, ACM, New York, NY, USA (1996)
2. Albrecht, M.R.: dgs — discrete gaussians over the integers. `https://bitbucket.org/malb/dgs` (2014)
3. Albrecht, M.R., Cocis, C., Laguillaumie, F., Langlois, A.: Implementing candidate graded encoding schemes from ideal lattices. In: Iwata, T., Cheon, J.H. (eds.)

ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 752–775. Springer, Heidelberg, Germany, Auckland, New Zealand (Nov 30 – Dec 3, 2015)
4. Babai, L.: On lovász' lattice reduction and the nearest lattice point problem. In: Mehlhorn, K. (ed.) STACS 85: 2nd Annual Symposium on Theoretical Aspects of Computer Science Saarbrücken, January 3–5, 1985, pp. 13–20. Springer Berlin Heidelberg, Berlin, Heidelberg (1985)
5. Bernstein, D.J.: The salsa20 family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs: The eSTREAM Finalists, pp. 84–97. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
6. Brent, R.P., et al.: Fast algorithms for high-precision computation of elementary functions. In: Proceedings of 7th conference on real numbers and computers (RNC 7). pp. 7–8 (2006)
7. Buchmann, J., Cabarcas, D., Göpfert, F., Hülsing, A., Weiden, P.: Discrete ziggurat: A time-memory trade-off for sampling from a Gaussian distribution over the integers. In: Lange, T., Lauter, K., Lisonek, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 402–417. Springer, Heidelberg, Germany, Burnaby, BC, Canada (Aug 14–16, 2014)
8. de Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Efficient software implementation of ring-lwe encryption. In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE). pp. 339–344 (2015)
9. Devroye, L.: Non-uniform random variate generation. Springer-Verlag (1986)
10. Ducas, L.: Lattice based signatures: Attacks, analysis and optimization. Ph.D. Thesis (2013)
11. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013)
12. Ducas, L., Nguyen, P.Q.: Faster Gaussian lattice sampling using lazy floating-point arithmetic. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 415–432. Springer, Heidelberg, Germany, Beijing, China (Dec 2–6, 2012)
13. Dwarakanath, N.C., Galbraith, S.D.: Sampling from discrete gaussians for lattice-based cryptography on a constrained device. Applicable Algebra in Engineering, Communication and Computing 25(3), 159–180 (2014)
14. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 1987)
15. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: Mpfr: A multiple-precision binary floating-point library with correct rounding. ACM Trans. Math. Softw. 33(2) (Jun 2007)
16. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing. pp. 197–206. STOC '08, ACM, New York, NY, USA (2008)
17. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press, Victoria, British Columbia, Canada (May 17–20, 2008)
18. Goldreich, O., Goldwasser, S., Halevi, S.: Public-key cryptosystems from lattice reduction problems. In: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology. pp. 112–131. CRYPTO '97, Springer-Verlag, London, UK, UK (1997)

19. Granlund, T., the GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library, 6.0.1 edn. (2015), `http://gmplib.org/`
20. Karney, C.F.F.: Sampling exactly from the normal distribution. ACM Trans. Math. Softw. 42(1), 3:1–3:14 (Jan 2016)
21. Klein, P.: Finding the closest lattice vector when it's unusually close. In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 937–941. SODA '00, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2000)
22. Knuth, D.E., Yao, A.C.: The Complexity of Nonuniform Random Number Generation. In: Traub, J.F. (ed.) Algorithms and Complexity: New Directions and Recent Results. Academic Press, New York (1976)
23. Lyubashevsky, V.: Lattice-based identification schemes secure under active attacks. In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 162–179. Springer, Heidelberg, Germany, Barcelona, Spain (Mar 9–12, 2008)
24. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg, Germany, Tokyo, Japan (Dec 6–10, 2009)
25. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval and Johansson [34], pp. 738–755
26. Lyubashevsky, V., Micciancio, D.: Asymptotically efficient lattice-based digital signatures. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 37–54. Springer, Heidelberg, Germany, San Francisco, CA, USA (Mar 19–21, 2008)
27. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg, Germany, French Riviera (May 30 – Jun 3, 2010)
28. Marsaglia, G., Tsang, W.W.: A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions. SIAM J. Sci. Stat. Comput. 5, 349–359 (1984)
29. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. Comput. Complex. 16(4), 365–411 (Dec 2007)
30. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on gaussian measures. SIAM J. Comput. 37(1), 267–302 (Apr 2007)
31. von Neumann, J.: Various Techniques Used in Connection with Random Digits. J. Res. Nat. Bur. Stand. 12, 36–38 (1951)
32. Peikert, C.: An efficient and parallel Gaussian sampler for lattices. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 80–97. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 2010)
33. Peikert, C.: A decade of lattice cryptography. Found. Trends Theor. Comput. Sci. 10(4), 283–424 (Mar 2016)
34. Pointcheval, D., Johansson, T. (eds.): EUROCRYPT 2012, LNCS, vol. 7237. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012)
35. Pujol, X., Stehlé, D.: Rigorous and efficient short lattice vectors enumeration. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 390–405. Springer, Heidelberg, Germany, Melbourne, Australia (Dec 7–11, 2008)
36. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing. pp. 84–93. STOC '05, ACM, New York, NY, USA (2005)
37. Saarinen, M.J.O.: Arithmetic coding and blinding countermeasures for lattice signatures. Journal of Cryptographic Engineering pp. 1–14 (2017)
38. Von Neumann, J.: The general and logical theory of automata. Cerebral mechanisms in behavior 1(41), 1–2 (1951)

39. Walker, A.J.: New fast method for generating discrete random numbers with arbitrary frequency distributions. Electronics Letters 10, 127–128(1) (April 1974)