Compact and Scalable Arbitrary-centered Discrete Gaussian Sampling over Integers

Raymond K. Zhao, Ron Steinfeld and Amin Sakzad

Faculty of Information Technology, Monash University,
{raymond.zhao,ron.steinfeld,amin.sakzad}@monash.edu

Abstract. The arbitrary-centered discrete Gaussian sampler is a fundamental subroutine in implementing lattice trapdoor sampling algorithms. However, existing approaches typically rely on either a fast implementation of another discrete Gaussian sampler or pre-computations with regards to some specific discrete Gaussian distributions with fixed centers and standard deviations. These approaches may only support sampling from standard deviations within a limited range, or cannot efficiently sample from arbitrary standard deviations determined on-the-fly at run-time.
In this paper, we propose a compact and scalable rejection sampling algorithm by sampling from a continuous normal distribution and performing rejection sampling on rounded samples. Our scheme does not require pre-computations related to any specific discrete Gaussian distributions. Our scheme can sample from both arbitrary centers and arbitrary standard deviations determined on-the-fly at run-time. In addition, we show that our scheme only requires a low number of trials close to 2 per sample on average, and our scheme maintains good performance when scaling up the standard deviation. We also provide a concrete error analysis of our scheme based on the Rényi divergence.

Keywords: Lattice-based crypto · discrete Gaussian sampling · implementation · efficiency

1 Introduction

The arbitrary-centered discrete Gaussian sampling algorithm is an important subroutine in implementing lattice trapdoor samplers, which is a fundamental tool employed by lattice-based cryptography applications such as digital signature [PFH+17] and identity-based encryption [GPV08, DLP14]. However, previous works focused more on optimising the lattice trapdoor sampling algorithms, but the implementation details of the arbitrary-centered discrete Gaussian sampling were not well addressed. Typically, arbitrary-centered discrete Gaussian sampling approaches need to perform either rejection sampling [DN12, Kar16, PFH+17, DWZ19, PRR19] or pre-computations related to some specific discrete Gaussian distributions [MAR17, MW17, MR18]. However, both types of methods have issues in the implementation: rejection sampling based methods are either slow due to the large number of trials per sample on average (typically, about 8–10) [DN12], requiring high precision arithmetic for cryptography applications [Kar16], or relying on a fast implementation of another discrete Gaussian sampler [PFH+17, DWZ19, PRR19]. On the other hand, pre-computation based methods consume at least few kilobytes (KB) of memory to store the tables and have the following limitations: the pre-computation table size in [MAR17, MR18] grows significantly when scaling up the standard deviation and this approach cannot support arbitrary standard deviations determined on-the-fly at run-time, while it is unclear how to efficiently implement the offline phase in [MW17] if the full algorithm needs to be executed during the run-time.

Recently the rounded Gaussian sampling (i.e. sampling from a continuous normal distribution and rounding the samples) was adapted by lattice-based digital signatures [ZCHW17, HLS18]. Compared with a previous discrete Gaussian sampling algorithm [DDLL13], the rounded Gaussian sampler in [HLS18] showed impressive performance with regards to the running speed and can be implemented in constant-time. The implementation in [HLS18] is also notably simple (within less than 40 lines of C++ source code). However, since it is unclear whether a rounded Gaussian distribution can be directly adapted to implement a lattice trapdoor, another interesting question is: can one employ the existing efficient (rounded) continuous Gaussian distribution sampling techniques to implement an arbitrary-centered discrete Gaussian sampler?

1.1 Contribution

In this paper, we introduce a novel arbitrary-centered discrete Gaussian sampling algorithm over integers by generalising ideas from [Dev86]. Our scheme samples from a continuous normal distribution and performs rejection sampling on rounded samples by adapting techniques from [ZCHW17, HLS18]. Compared to previous arbitrary-centered discrete Gaussian sampling techniques, our scheme has the following advantages:

- Our sampling algorithm does not require any pre-computations related to a specific discrete Gaussian distribution or a specific standard deviation, and both the center and the standard deviation can be arbitrary determined on-the-fly at run-time.
- In addition, we show in Section 4 that our sampling method only requires a low number of trials close to 2 per sample on average compared to about 8–10 on average in the rejection sampling with regards to a uniform distribution, and the rejection rate of our algorithm decreases when scaling up σ. Therefore, our sampling algorithm is not limited to small σ and can be adapted to sample from larger σ without affecting the efficiency.
- Since sampling from a continuous normal distribution is a well-studied topic [TLLV07] and the sampling algorithms are implemented in many existing software libraries (including the C++11 STL) and hardware devices, one can easily implement our scheme by employing existing tools.

2 Preliminaries

Let $\rho_{c,\sigma}(x) = \exp\left(-(x-c)^2/2\sigma^2\right)$ be the (continuous) Gaussian function with center c and standard deviation σ. We denote the continuous Gaussian (normal) distribution with center c and standard deviation σ by $\mathcal{N}(c, \sigma^2)$, which has the probability density function $\rho_{c,\sigma}(x)/\left(\sigma\sqrt{2\pi}\right)$. We denote the discrete Gaussian distribution on integer lattices with center c and standard deviation σ by: $\mathcal{D}_{c,\sigma}(x) = \rho_{c,\sigma}(x)/S$, where $S = \rho_{c,\sigma}(\mathbb{Z}) = \sum_{k\in\mathbb{Z}} \rho_{c,\sigma}(k)$ is the normalisation factor. We omit the center in notations (i.e. $\rho_\sigma(x)$ and $\mathcal{D}_\sigma(x)$) if the center is zero. In addition, we denote the uniform distribution on set S by $\mathcal{U}(S)$. Sampling from a distribution $\mathcal{P}$ is denoted by $x \hookleftarrow \mathcal{P}$. We define $\lfloor x \rceil$ as the nearest integer to $x \in \mathbb{R}$. We denote $\mathbb{Z}^+$ as the integer set $\{1, \ldots, \infty\}$ and $\mathbb{Z}^-$ as the integer set $\{-\infty, \ldots, -1\}$, respectively. Also, for a lattice Λ and any $\epsilon \in \mathbb{R}^+$, we denote the smoothing parameter $\eta_\epsilon(\Lambda)$ as the smallest $s \in \mathbb{R}^+$ such that $\rho_{1/(s\sqrt{2\pi})}\left(\Lambda^* \setminus \{\vec{0}\}\right) \leq \epsilon$, where Λ^* is the dual lattice of Λ: $\Lambda^* = \{\vec{w} \in \mathbb{R}^n : \forall \vec{x} \in \Lambda, \vec{x}\cdot\vec{w} \in \mathbb{Z}\}$ [Pei10]. An upper bound on $\eta_\epsilon(\mathbb{Z})$ is given by [Pei10]: $\eta_\epsilon(\mathbb{Z}) \leq \sqrt{\ln(2+2/\epsilon)/\pi}$.

Theorem 1 (Adapted from [Pei10], Lemma 2.4)**.** *For any $\epsilon \in (0,1)$ and $c \in \mathbb{R}$, if $\sigma \geq \eta_\epsilon(\mathbb{Z})$, then $\rho_{c,\sigma}(\mathbb{Z}) = [(1-\epsilon)/(1+\epsilon), 1] \cdot \rho_\sigma(\mathbb{Z})$, and $\rho_\sigma(\mathbb{Z})$ is approximately $\int_{-\infty}^{\infty} \rho_\sigma(x)\,\mathrm{d}x = \sigma\sqrt{2\pi}$.*

Definition 1 (Relative Error)**.** For two distributions $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathrm{Supp}(\mathcal{P}) = \mathrm{Supp}(\mathcal{Q})$, the relative error between $\mathcal{P}$ and $\mathcal{Q}$ is defined as:

$$\Delta(\mathcal{P}||\mathcal{Q}) = \max_{x \in \mathrm{Supp}(\mathcal{P})} \frac{|\mathcal{P}(x) - \mathcal{Q}(x)|}{\mathcal{Q}(x)}.$$

Definition 2 (Rényi Divergence [BLL+15, Pre17])**.** For two discrete distributions $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathrm{Supp}(\mathcal{P}) \subseteq \mathrm{Supp}(\mathcal{Q})$, the Rényi divergence (RD) of order $\alpha \in (1, +\infty)$ is defined as:

$$R_\alpha(\mathcal{P}||\mathcal{Q}) = \left(\sum_{x \in \mathrm{Supp}(\mathcal{P})} \frac{\mathcal{P}(x)^\alpha}{\mathcal{Q}(x)^{\alpha-1}} \right)^{\frac{1}{\alpha-1}}.$$

Theorem 2 (Relative Error Bound, Adapted from [Pre17], Lemma 3 and Eq. 4)**.** *For two distributions $\mathcal{P}$ and $\mathcal{Q}$ such that* $\mathrm{Supp}(\mathcal{P}) = \mathrm{Supp}(\mathcal{Q})$*, we have:*

$$R_\alpha(\mathcal{P}||\mathcal{Q}) \leq \left(1 + \frac{\alpha(\alpha-1) \cdot (\Delta(\mathcal{P}||\mathcal{Q}))^2}{2(1 - \Delta(\mathcal{P}||\mathcal{Q}))^{\alpha+1}} \right)^{\frac{1}{\alpha-1}}.$$

The right-hand side is asymptotically equivalent to $1 + \alpha \cdot (\Delta(\mathcal{P}||\mathcal{Q}))^2/2$ *as* $\Delta(\mathcal{P}||\mathcal{Q}) \to 0$*. In addition, if a cryptographic search problem using M independent samples from $\mathcal{Q}$ is $(\lambda+1)$-bit secure, then the same problem sampling from $\mathcal{P}$ will be λ-bit secure if* $R_{2\lambda}(\mathcal{P}||\mathcal{Q}) \leq 1 + 1/(4M)$*.*

3 Previous Work

In this section, we review previous techniques of sampling from $\mathcal{D}_{c,\sigma}$ with an arbitrary center c.

3.1 Rejection Sampling

The classic rejection sampling algorithm [von51, DN12] can sample from an arbitrary-centered discrete Gaussian distribution. To sample from $\mathcal{D}_{c,\sigma}$, one can sample $x \hookleftarrow \mathcal{U}([c - \tau\sigma, c + \tau\sigma] \cap \mathbb{Z})$ and accept x with probability $\rho_{c,\sigma}(x)$ as the output, where τ is the tail-cut factor (typically, about 10–12). However, this method is slow as the number of trials is $2\tau/\sqrt{2\pi}$ on average (about 8–10 for typical τ). Recently an algorithm sampling exactly from $\mathcal{D}_{c,\sigma}$ without floating-point arithmetic was presented by [Kar16], which also has a lower rejection rate compared to the classic rejection sampling algorithm. However, this algorithm relies on high precision integer arithmetic to satisfy the precision requirements in cryptography applications.

To reduce the rejection rate, recent works performed rejection sampling with regards to some distributions much closer to $\mathcal{D}_{c,\sigma}$ compared to a uniform distribution:

- The Falcon signature [PFH+17] and its constant-time variant [PRR19] adapted a rejection sampling method with regards to bimodal Gaussians: to sample from $\mathcal{D}_{c,\sigma}$ where $c \in [0, 1]$, one can choose some $\sigma' \geq \sigma$ and sample $x \hookleftarrow \mathcal{D}^+_{\sigma'}$ (i.e. the discrete Gaussian distribution $\mathcal{D}_{\sigma'}$ restricted to the domain $\mathbb{Z}^+ \cup \{0\}$). The algorithm computes $x' = b + (2b-1) \cdot x$ where $b \hookleftarrow \mathcal{U}(\{0, 1\})$. The authors of [PFH+17, PRR19] showed that x' has a bimodal Gaussian distribution close to the target distribution. The algorithm then accepts x' with probability $C(\sigma) \cdot \exp\left(\left(x^2/\left(2\sigma'^2\right)\right) - \left((x'-c)^2/\left(2\sigma^2\right)\right) \right)$ as the output, where the scaling factor $C(\sigma) = \min(\sigma)/\sigma$ when sampling from multiple σ. This scheme has the average

acceptance rate $C(\sigma) \cdot \rho_{c,\sigma}(\mathbb{Z}) / (2\rho_{\sigma'}(\mathbb{Z}^+))$, which is proportional to $\min(\sigma)/\sigma'$ [PFH+17, PRR19]. However, if the application needs to sample from different σ, the acceptance probability is high only when $\min(\sigma)$ and $\max(\sigma)$ are sufficiently close. This is not an issue in the Falcon signature, since the parameters in Falcon implies σ' is very close to $\max(\sigma)$ and $\min(\sigma)/\max(\sigma) \approx 0.73$ [PRR19]. However, if the gap between $\min(\sigma)$ and $\max(\sigma)$ is large, since $\sigma' \geq \max(\sigma)$, this algorithm might have a low acceptance rate[1].

- A recent work [DWZ19] extended the binary sampling algorithm from the BLISS signature [DDLL13] to support non-zero arbitrary centers. For any center $c \in \mathbb{R}$, sampling from $\mathcal{D}_{c,\sigma}$ is equivalent to sampling from $\mathcal{D}_{c_F,\sigma} + \lfloor c \rfloor$, where $c_F = c - \lfloor c \rfloor \in [0, 1)$ is the fractional part of c. In addition, for $1/2 \leq c_F < 1$, sampling from $\mathcal{D}_{c_F,\sigma}$ is equivalent to sampling from $1 - \mathcal{D}_{c'_F,\sigma}$ where $c'_F = 1 - c_F \in (0, 1/2]$. A modified binary sampling scheme [DWZ19] can then be adapted to sample from $\mathcal{D}_{c'_F,\sigma}$ with any $c'_F \in (0, 1/2]$, in which the average number of trials is upper-bounded by: $\left(\sigma^2 / \left(\sigma_0\sigma - \sigma_0^2\right)\right) \cdot \left(\rho_{\sigma_0}(\mathbb{Z}^+) / \left(\sigma\sqrt{\pi/2} - 1\right)\right)$, where $\sigma_0 = \sqrt{1/(2\ln 2)}$ is a fixed parameter used by the binary sampling algorithm [DDLL13, DWZ19] and $\sigma = k\sigma_0$ for some $k \in \mathbb{Z}^+$. This upper-bound is about 1.47 for large σ [DWZ19].

3.2 TwinCDT

The authors of [MAR17, MR18] suggested a variant of the Cumulative Distribution Table (CDT) method [Dev86] with multiple pre-computed tables. These algorithms will have two phases: online and offline. To be more specific, for $c \in [0, 1)$, during the offline phase, the algorithm pre-computes multiple CDT of $\mathcal{D}_{i/n,\sigma}$, where $i \in \{0, \ldots, n-1\}$ and $n \in \mathbb{Z}^+$ is sufficiently large. During the online phase, the algorithm picks a sample generated from either $\mathcal{D}_{\lfloor n(c-\lfloor c \rfloor)\rfloor/n,\sigma}$ or $\mathcal{D}_{\lceil n(c-\lfloor c \rfloor)\rceil/n,\sigma}$ as the output. Although the algorithm is very fast compared to other approaches, however, σ is fixed during the offline computation and thus this algorithm cannot support sampling from $\mathcal{D}_{c,\sigma}$ with both arbitrary c and σ determined on-the-fly at run-time. Another issue is that the pre-computation table size grows significantly when scaling up σ (see Table 2 in Section 5) and therefore the algorithm is not scalable.

3.3 Convolution

A recursive convolution sampling scheme for $\mathcal{D}_{c,\sigma}$ was presented in [MW17] as follows: suppose the center c has k fractional bits. Let $\sigma_0 = \sigma / \sqrt{\sum_{i=0}^{k-1} 2^{-2i}}$. One can sample $x_k \hookleftarrow \mathcal{D}_{c_k,\sigma_0}$ where $c_k = 2^{k-1} \cdot c$, then use $y_k = 2^{-k+1} \cdot x_k$ to round c to a new center $c' = c - y_k$ with $k' = k - 1$ fractional bits. Set $c = c'$ and $k = k'$ in the next iteration until $k = 0$, and $\sum_{i=1}^{k} y_i$ will be a sample distributed as $\mathcal{D}_{c,\sigma}$. The authors of [MW17] separated this algorithm into an online phase and an offline phase, where the offline phase will generate samples x_i in batch and the online phase will compute the linear combinations of x_i for $1 \leq i \leq k$. The online phase is very fast and can be implemented in constant-time. However, for implementations where both sampling from $\mathcal{D}_{c_i,\sigma_0}$ and computing the linear combinations need to be carried during the run-time, it is unclear how to efficiently implement the $\mathcal{D}_{c_i,\sigma_0}$ sampling algorithm in constant-time (which is another discrete Gaussian sampler supporting a small amount of centers c_i). The offline batch sampler also consumes significant amount of memory (see Table 2 in Section 5).

[1] One may employ different implementations for different σ, similar to the implementation of Falcon.

Algorithm 1 Adapted from [Dev86], pg. 117, ch. 3

Input: Standard deviation $\sigma \in \mathbb{R}^+$.

Output: A sample z distributed as $\Pr[X = z] = c \cdot \exp\left(-\left(|z| + 1/2\right)^2 / \left(2\sigma^2\right)\right)$.

1: **function** SAMPLER(σ)
2: Sample $x \hookleftarrow \mathcal{N}\left(0, \sigma^2\right)$.
3: Sample $r \hookleftarrow \mathcal{U}\left([0, 1)\right)$.
4: Let $Y = \left(|\lfloor x \rceil| + 1/2\right)^2 - x^2$.
5: **if** $r < \exp\left(-Y / \left(2\sigma^2\right)\right)$ **then**
6: Let $z = \lfloor x \rceil$.
7: **else**
8: **goto** 2.
9: **end if**
10: **return** z.
11: **end function**

4 Proposed Algorithm

In the textbook [Dev86], the author defined a variant of the discrete Gaussian distribution as $\Pr[X = z] = c \cdot \exp\left(-\left(|z| + 1/2\right)^2 / \left(2\sigma^2\right)\right)$, where $z \in \mathbb{Z}$ and c is the normalisation constant, i.e.

$$\Pr[X = z] \propto \begin{cases} \rho_{-1/2,\sigma}(z) & z \geq 0, \\ \rho_{1/2,\sigma}(z) & z < 0. \end{cases}$$

A rejection sampling algorithm (see Algorithm 1) was provided by [Dev86] with rejection probability less than $(2/\sigma) \cdot \sqrt{2/\pi}$ for such a distribution, which is fast for large σ.

Here we generalise Algorithm 1 to sample from $\mathcal{D}_{c,\sigma}(z)$. By removing the absolute value and replacing the fixed center $-1/2$ with a generic center c in Algorithm 1, we observe that $Y' = \left(\lfloor x \rceil + c\right)^2 - x^2 \geq 0$ when $(c \geq 1/2, x \geq 0)$ or $(c \leq -1/2, x < 0)$. Therefore, we can replace Y with Y' and perform a similar rejection sampling to Algorithm 1 when sampling from $\mathcal{D}_{c,\sigma}(z)$ for some c and $z = \lfloor x \rceil$. To extend Algorithm 1 to support all $c \in \mathbb{R}$ and $z \in \mathbb{Z}$, we first compute $c_I = \lfloor c \rceil$ and $c_F = c_I - c$, where $c_F \in [-1/2, 1/2]$. Then we can sample from $\mathcal{D}_{-c_F,\sigma}$ instead, since $\mathcal{D}_{c,\sigma} = \mathcal{D}_{-c_F,\sigma} + c_I$. To sample from $\mathcal{D}_{-c_F,\sigma}$ for all $c_F \in [-1/2, 1/2]$, we shift the center of the underlying continuous normal distribution, i.e. sampling $y \hookleftarrow \mathcal{N}\left(\pm 1, \sigma^2\right)$, and perform a rejection sampling over $z = \lfloor y \rceil$ with acceptance rate $\exp\left(-Y'' / \left(2\sigma^2\right)\right)$ where $Y'' = \left(\lfloor y \rceil + c_F\right)^2 - \left(y \mp 1\right)^2$ (we also need to ensure $Y'' \geq 0$ before performing this rejection sampling). The sampling algorithm for $\mathcal{D}_{-c_F,\sigma}$ is presented in Algorithm 2. Note that the output of Algorithm 2 is restricted to the domain $\mathbb{Z} \setminus \{0\}$. Therefore, the algorithm needs to output 0 with probability $\mathcal{D}_{-c_F,\sigma}(0)$. We present the full algorithm in Algorithm 3. Since both Algorithm 2 and Algorithm 3 do not require pre-computations related to σ, our scheme can support arbitrary standard deviations determined on-the-fly at run-time in addition to arbitrary centers.

Theorem 3. *The output z sampled by Algorithm 2 is distributed as $\mathcal{D}_{-c_F,\sigma}\left(\mathbb{Z} \setminus \{0\}\right)$. The output of Algorithm 3 is distributed as $\mathcal{D}_{c,\sigma}(\mathbb{Z})$.*

Proof. When $b = 0$, y is distributed as $\mathcal{N}\left(-1, \sigma^2\right)$. For step 11 in Algorithm 2, we have $Y_1 = \left(\lfloor y \rceil + c_F\right)^2 - \left(y + 1\right)^2 \geq 0$ for any $c_F \in [-1/2, 1/2]$ when $y \leq -1/2$. Therefore, the rejection condition $\exp\left(-Y_1 / \left(2\sigma^2\right)\right) \in (0, 1]$. Let $z_0 = \lfloor y \rceil$. We have the output

Algorithm 2 $\mathcal{D}_{-c_F,\sigma}(\mathbb{Z} \setminus \{0\})$ sampler

Input: Center $c_F \in [-1/2, 1/2]$. Standard deviation $\sigma \in \mathbb{R}^+$.
Output: A sample z distributed as $\mathcal{D}_{-c_F,\sigma}$ restricted to the domain $\mathbb{Z} \setminus \{0\}$.

function RoundingSampler(c_F, σ)
  Sample $x \hookleftarrow \mathcal{N}(0,1)$.
  Sample $b \hookleftarrow \mathcal{U}(\{0,1\})$.
  if $b = 0$ **then**
    Let $y = \sigma \cdot x - 1$.
    if $y > -1/2$ **then**
      goto 2.
    end if
    Sample $r \hookleftarrow \mathcal{U}([0,1))$.
    Let $Y_1 = (\lfloor y \rceil + c_F)^2 - (y+1)^2$.
    if $r < \exp\left(-Y_1/\left(2\sigma^2\right)\right)$ **then**
      Let $z = \lfloor y \rceil$.
    else
      goto 2.
    end if
  else
    Let $y = \sigma \cdot x + 1$.
    if $y < 1/2$ **then**
      goto 2.
    end if
    Sample $r \hookleftarrow \mathcal{U}([0,1))$.
    Let $Y_2 = (\lfloor y \rceil + c_F)^2 - (y-1)^2$.
    if $r < \exp\left(-Y_2/\left(2\sigma^2\right)\right)$ **then**
      Let $z = \lfloor y \rceil$.
    else
      goto 2.
    end if
  end if
  return z.
end function

Algorithm 3 $\mathcal{D}_{c,\sigma}(\mathbb{Z})$ sampler

Input: Center $c \in \mathbb{R}$. Standard deviation $\sigma \in \mathbb{R}^+$. Normalisation factor $S = \rho_{c,\sigma}(\mathbb{Z}) \approx \sigma\sqrt{2\pi}$.
Output: A sample distributed as $\mathcal{D}_{c,\sigma}(\mathbb{Z})$.

function RoundingSamplerFull(c, σ)
  Let $c_I = \lfloor c \rceil$ and $c_F = c_I - c$.
  Sample $r \hookleftarrow \mathcal{U}([0,1))$.
  if $r < \exp\left(-c_F^2/\left(2\sigma^2\right)\right)/S$ **then**
    Let $z' = 0$.
  else
    Let $z' = \text{RoundingSampler}(c_F, \sigma)$.
  end if
  return $z' + c_I$.
end function

distribution:

$$\begin{aligned}\Pr[z = z_0] &\propto \int_{z_0-1/2}^{z_0+1/2} \exp\left(-\frac{(y+1)^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{(z_0+c_F)^2 - (y+1)^2}{2\sigma^2}\right) \mathrm{d}y \\ &= \int_{z_0-1/2}^{z_0+1/2} \exp\left(-\frac{(z_0+c_F)^2}{2\sigma^2}\right) \mathrm{d}y \\ &= \rho_{-c_F,\sigma}(z_0).\end{aligned} \tag{1}$$

In this case, the distribution of $z = z_0$ is $\mathcal{D}_{-c_F,\sigma}$ restricted to the domain $\mathbb{Z}^-$ (due to the rejection of y to $(-\infty, -1/2]$).

Similarly, when $b = 1$, y is distributed as $\mathcal{N}\left(1, \sigma^2\right)$. For step 23 in Algorithm 2, we have $Y_2 = (\lfloor y \rceil + c_F)^2 - (y-1)^2 \geq 0$ for any $c_F \in [-1/2, 1/2]$ when $y \geq 1/2$. Therefore, the rejection condition $\exp\left(-Y_2/\left(2\sigma^2\right)\right) \in (0, 1]$. Let $z_0 = \lfloor y \rceil$. We have the output distribution:

$$\begin{aligned}\Pr[z = z_0] &\propto \int_{z_0-1/2}^{z_0+1/2} \exp\left(-\frac{(y-1)^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{(z_0+c_F)^2 - (y-1)^2}{2\sigma^2}\right) \mathrm{d}y \\ &= \int_{z_0-1/2}^{z_0+1/2} \exp\left(-\frac{(z_0+c_F)^2}{2\sigma^2}\right) \mathrm{d}y \\ &= \rho_{-c_F,\sigma}(z_0).\end{aligned} \tag{2}$$

In this case, the distribution of $z = z_0$ is $\mathcal{D}_{-c_F,\sigma}$ restricted to the domain $\mathbb{Z}^+$ (due to the rejection of y to $[1/2, \infty)$). Therefore, the output z in Algorithm 2 is distributed as $\mathcal{D}_{-c_F,\sigma}$ restricted to the domain $\mathbb{Z} \setminus \{0\}$.

In Algorithm 3, the probability $\Pr[z' = 0] = \exp\left(-c_F^2/\left(2\sigma^2\right)\right)/S = \mathcal{D}_{-c_F,\sigma}(0)$. Therefore, variable z' is distributed as $\mathcal{D}_{-c_F,\sigma}(\mathbb{Z})$. Since $c = c_I - c_F$, we have the output $z' + c_I$ distributed as $\mathcal{D}_{c,\sigma}(\mathbb{Z})$. □

Theorem 4. *For $\sigma \geq \eta_\epsilon(\mathbb{Z})$, the expected number of trials M in Algorithm 2 has the upper bound:*

$$M \leq \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\sigma\sqrt{\pi/2} - 1}.$$

If σ is much greater than $\sqrt{2/\pi}$, then $M \leq 2 + O(\epsilon)$.

Proof. By Theorem 3, when $b = 0$, we have the output probability density function $f(y) = \rho_{-c_F,\sigma}(\lfloor y \rceil)/\rho_{-c_F,\sigma}(\mathbb{Z}^-)$ and the input probability density function $g(y) = \rho_{-1,\sigma}(y)/\left(\sigma\sqrt{2\pi}\right)$. The expected number of trials can be written as:

$$M = \max \frac{f(y)}{g(y)} = \max\left(\frac{\rho_{-c_F,\sigma}(\lfloor y \rceil)}{\rho_{-1,\sigma}(y)} \cdot \frac{\sigma\sqrt{2\pi}}{\rho_{-c_F,\sigma}(\mathbb{Z}^-)}\right).$$

We have:

$$\frac{\rho_{-c_F,\sigma}(\lfloor y \rceil)}{\rho_{-1,\sigma}(y)} = \frac{\exp\left(-(\lfloor y \rceil + c_F)^2/\left(2\sigma^2\right)\right)}{\exp\left(-(y+1)^2/(2\sigma^2)\right)} = \exp\left(-\frac{(\lfloor y \rceil + c_F)^2 - (y+1)^2}{2\sigma^2}\right) \leq 1.$$

Therefore,

$$M \leq \frac{\sigma\sqrt{2\pi}}{\rho_{-c_F,\sigma}(\mathbb{Z}^-)} \leq \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\rho_\sigma(\mathbb{Z}^-)} \leq \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\sigma\sqrt{\pi/2} - 1},$$

where the second inequality follows from Theorem 1, and the third inequality follows from $\rho_\sigma(\mathbb{Z}^-) = \rho_\sigma(\mathbb{Z}^- \cup \{0\}) - 1$ and the sum-integral comparison:

$$\rho_\sigma\left(\mathbb{Z}^- \cup \{0\}\right) \geq \int_{-\infty}^{0} \rho_\sigma(x)\,\mathrm{d}x = \sigma\sqrt{\pi/2}.$$

Similarly, when $b = 1$, we have the output probability density function $f(y) = \rho_{-c_F,\sigma}(\lfloor y \rceil)/\rho_{-c_F,\sigma}(\mathbb{Z}^+)$ and the input probability density function $g(y) = \rho_{1,\sigma}(y)/\left(\sigma\sqrt{2\pi}\right)$. The expected number of trials can be written as:

$$M = \max \frac{f(y)}{g(y)} = \max \left(\frac{\rho_{-c_F,\sigma}(\lfloor y \rceil)}{\rho_{1,\sigma}(y)} \cdot \frac{\sigma\sqrt{2\pi}}{\rho_{-c_F,\sigma}(\mathbb{Z}^+)} \right).$$

We have:

$$\frac{\rho_{-c_F,\sigma}(\lfloor y \rceil)}{\rho_{1,\sigma}(y)} = \frac{\exp\left(-(\lfloor y \rceil + c_F)^2 / (2\sigma^2)\right)}{\exp\left(-(y-1)^2/(2\sigma^2)\right)} = \exp\left(-\frac{(\lfloor y \rceil + c_F)^2 - (y-1)^2}{2\sigma^2}\right) \leq 1.$$

Therefore,

$$M \leq \frac{\sigma\sqrt{2\pi}}{\rho_{-c_F,\sigma}(\mathbb{Z}^+)} \leq \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\rho_{\sigma}(\mathbb{Z}^+)} \leq \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\sigma\sqrt{\pi/2}-1},$$

where the second inequality follows from Theorem 1, and the third inequality follows from $\rho_\sigma(\mathbb{Z}^+) = \rho_\sigma(\mathbb{Z}^+ \cup \{0\}) - 1$ and the sum-integral comparison:

$$\rho_\sigma\left(\mathbb{Z}^+ \cup \{0\}\right) \geq \int_0^\infty \rho_\sigma(x)\,\mathrm{d}x = \sigma\sqrt{\pi/2}.$$

When σ is much greater than $\sqrt{2/\pi}$, $\sigma\sqrt{\pi/2}$ is much greater than 1. Thus,

$$M \leq \frac{1+\epsilon}{1-\epsilon} \cdot \frac{\sigma\sqrt{2\pi}}{\sigma\sqrt{\pi/2}-1} = 2 \cdot \frac{\sigma\sqrt{\pi/2}}{\sigma\sqrt{\pi/2}-1} \cdot \frac{1+\epsilon}{1-\epsilon} \leq 2 + O(\epsilon).$$

□

We now analyse the relative error of Algorithm 2 here. Let the absolute error of the continuous Gaussian sample x be e_x: $x' = x + e$, where x' is the actual sample, x is the ideal sample, and the error $|e| \leq e_x$. We denote the actual distribution by $\mathcal{P}_{\text{actual}}$ and the ideal distribution by $\mathcal{P}_{\text{ideal}}$. Since the variable y might be rounded to an incorrect integer due to the error from x when y is close to the boundaries $z_0 \pm 1/2$ [HLS18], we have:

$$\begin{aligned}
&\Delta(\mathcal{P}_{\text{actual}}||\mathcal{P}_{\text{ideal}}) = \max \left| \frac{\mathcal{P}_{\text{actual}}}{\mathcal{P}_{\text{ideal}}} - 1 \right| \\
&= \max_{z_0} \left| \frac{\int_{z_0-1/2-\sigma e_x}^{z_0+1/2+\sigma e_x} \exp\left(-(z_0+c_F)^2/(2\sigma^2)\right)\,\mathrm{d}y}{\rho_{-c_F,\sigma}(z_0)} - 1 \right| \quad \text{(by (1), (2), and } y = \sigma x \pm 1) \\
&= \max_{z_0} \left| \frac{(1+2\sigma e_x)\cdot \rho_{-c_F,\sigma}(z_0)}{\rho_{-c_F,\sigma}(z_0)} - 1 \right| \\
&= 2\sigma e_x.
\end{aligned}$$

By Theorem 2, for λ-bit security, we need:

$$R_{2\lambda}(\mathcal{P}_{\text{actual}}||\mathcal{P}_{\text{ideal}}) \leq 1 + \frac{1}{4M} \implies 1 + 2\lambda \cdot \frac{(\Delta(\mathcal{P}_{\text{actual}}||\mathcal{P}_{\text{ideal}}))^2}{2} \leq 1 + \frac{1}{4M} \implies e_x \leq \frac{1}{4\sigma\sqrt{\lambda M}}.$$

4.1 Precision Analysis

To avoid sampling a uniformly random real r with high absolute precisions at rejection steps 11 and 23 in Algorithm 2, and step 4 in Algorithm 3, we adapt the comparison approach similar to [ZSS18]. Assume an IEEE-754 floating-point value $f \in (0,1)$ with (δ_f+1)-bit precision is represented by $f = \left(1 + mantissa \cdot 2^{-\delta_f}\right) \cdot 2^{exponent}$, where integer $mantissa$ has δ_f bits and $exponent \in \mathbb{Z}^-$. To check $r < f$, one can sample $r_m \hookleftarrow \mathcal{U}\left(\{0,1\}^{\delta_f+1}\right)$, $r_e \hookleftarrow \mathcal{U}\left(\{0,1\}^l\right)$, and check $r_m < mantissa + 2^{\delta_f}$ and $r_e < 2^{l+exponent+1}$ instead for some l such that $l + exponent + 1 \geq 0$.

Here we analyse the precision requirement of r_e. We have the following theorem for the worst-case acceptance rate in Algorithm 2:

Theorem 5. *Assume $x \in [-\tau, \tau]$ and $y \in [-\tau\sigma - 1, \tau\sigma + 1]$. In worst case, step 11 in Algorithm 2 has the acceptance rate:*

$$p_1 \geq \exp\left(-\frac{(-2\tau\sigma + c_F - 3/2)(c_F - 3/2)}{2\sigma^2}\right),$$

and step 23 in Algorithm 2 has the acceptance rate:

$$p_2 \geq \exp\left(-\frac{(2\tau\sigma + c_F + 3/2)(c_F + 3/2)}{2\sigma^2}\right).$$

Proof. For $b = 0$ and $y \leq -1/2$, we have the acceptance rate $p_1 = \exp\left(-Y_1/\left(2\sigma^2\right)\right)$ at step 11 in Algorithm 2 where:

$$\begin{aligned} Y_1 &= (\lfloor y \rceil + c_F)^2 - (y+1)^2 \\ &= (y + \delta + c_F)^2 - (y+1)^2 \quad (\lfloor y \rceil = y + \delta \text{ where } \delta \in [-1/2, 1/2]) \\ &= (2y + \delta + c_F + 1)(\delta + c_F - 1) \\ &\leq (-2\tau\sigma + c_F - 3/2)(c_F - 3/2). \quad (\text{when } \delta = -1/2 \text{ and } y = -\tau\sigma - 1) \end{aligned}$$

Similarly, for $b = 1$ and $y \geq 1/2$, we have the acceptance rate $p_2 = \exp\left(-Y_2/\left(2\sigma^2\right)\right)$ at step 23 in Algorithm 2 where:

$$\begin{aligned} Y_2 &= (\lfloor y \rceil + c_F)^2 - (y-1)^2 \\ &= (y + \delta + c_F)^2 - (y-1)^2 \quad (\lfloor y \rceil = y + \delta \text{ where } \delta \in [-1/2, 1/2]) \\ &= (2y + \delta + c_F - 1)(\delta + c_F + 1) \\ &\leq (2\tau\sigma + c_F + 3/2)(c_F + 3/2). \quad (\text{when } \delta = 1/2 \text{ and } y = \tau\sigma + 1) \end{aligned}$$

□

Let $\Delta \leq 1/2$ be the maximum relative error of the right hand side computations at rejection steps 11 and 23 in Algorithm 2, and step 4 in Algorithm 3. For $\exp\left(-Y_1/\left(2\sigma^2\right)\right)$ at step 11 in Algorithm 2, we have:

$$\begin{aligned} \mathit{exponent}_1 &\geq \left\lfloor \log_2\left((1-\Delta)\cdot\exp\left(-\frac{Y_1}{2\sigma^2}\right)\right)\right\rfloor \\ &\geq \left\lfloor -1 - \frac{(-2\tau\sigma + c_F - 3/2)(c_F - 3/2)}{2\sigma^2}\cdot\log_2 e\right\rfloor \quad (\text{by Theorem 5 and } \Delta \leq 1/2) \\ &\geq \left\lfloor -1 - \frac{2\tau\sigma + 2}{\sigma^2}\cdot\log_2 e\right\rfloor. \quad (\text{when } c_F = -1/2) \end{aligned}$$

Similarly, for $\exp\left(-Y_2/\left(2\sigma^2\right)\right)$ at step 23 in Algorithm 2, we have:

$$\begin{aligned} \mathit{exponent}_2 &\geq \left\lfloor \log_2\left((1-\Delta)\cdot\exp\left(-\frac{Y_2}{2\sigma^2}\right)\right)\right\rfloor \\ &\geq \left\lfloor -1 - \frac{(2\tau\sigma + c_F + 3/2)(c_F + 3/2)}{2\sigma^2}\cdot\log_2 e\right\rfloor \quad (\text{by Theorem 5 and } \Delta \leq 1/2) \\ &\geq \left\lfloor -1 - \frac{2\tau\sigma + 2}{\sigma^2}\cdot\log_2 e\right\rfloor. \quad (\text{when } c_F = 1/2) \end{aligned}$$

For $\exp\left(-c_F^2/\left(2\sigma^2\right)\right)/S$ at step 4 in Algorithm 3, we have:

$$\begin{aligned} exponent_3 &\geq \left\lfloor \log_2\left((1-\Delta)\cdot\exp\left(-\frac{c_F^2}{2\sigma^2}\right)/S\right)\right\rfloor \\ &\geq \left\lfloor -1-\frac{1}{8\sigma^2}\cdot\log_2 e-\log_2\left(\sigma\sqrt{2\pi}\right)\right\rfloor. \quad \text{(when } c_F=\pm 1/2 \text{ and } \Delta\leq 1/2\text{)} \end{aligned}$$

Therefore, we have:

$$exponent \geq \min\left\{\left\lfloor -1-\frac{2\tau\sigma+2}{\sigma^2}\cdot\log_2 e\right\rfloor, \left\lfloor -1-\frac{1}{8\sigma^2}\cdot\log_2 e-\log_2\left(\sigma\sqrt{2\pi}\right)\right\rfloor\right\}.$$

Since the probability $\Pr\left[-\tau\leq x\leq\tau\right]=\operatorname{erf}\left(\tau/\sqrt{2}\right)$ for $x\hookleftarrow\mathcal{N}(0,1)$, to ensure $1-\Pr\left[-\tau\leq x\leq\tau\right]\leq 2^{-\lambda}$, we need $\tau\geq\sqrt{2}\cdot\operatorname{erf}^{-1}\left(1-2^{-\lambda}\right)$. Therefore, for $\lambda=128$ and $\sigma\in\left[2,2^{15}\right]$, we have $\tau\geq 13.11$, $exponent\geq -21$, and thus $l\geq 20$, i.e. r_e needs to have at least 20 bits.

5 Performance

We perform benchmarks of Algorithm 3 with fixed σ and random arbitrary centers. The scheme[2] is implemented by using the double precision i.e. $\delta_f=52$. We employ the Box-Muller continuous Gaussian sampler [ZCHW17, HLS18] implemented by using the VCL library [Fog], which provides $e_x\leq 2^{48}$ [HLS18]. We use the AES256 counter mode with hardware AES instructions (AES-NI) to generate the randomness in our implementations. We provide both the non-constant time reference implementation and the constant-time implementation (note that the rejection rate in the constant-time implementation may still reveal σ due to Theorem 4). For the non-constant time reference implementation, we use the $\exp(x)$ from the C library, which provides about 50-bit precision [PFH+17], while for the constant-time implementation, we adapt the techniques from [ZSS18] with about 45-bit precision. To compare with [MR18], we select $\sigma=\{2,4,8,16,32\}$, and to compare with [MW17, DWZ19], we choose $\sigma=2^{15}$. From the error analysis in Section 4, for $\sigma\in\left[2,2^{15}\right]$ and $\lambda=128$, we have $M\leq 2^{54}$.

The benchmark is carried on as follows: we use `g++` 9.1.1 to compile our implementations with the compiling options `-O3 -march=native` enabled. The benchmark is running on an Intel i7-7700K CPU at 4.2GHz, with the Hyperthreading and the Turbo Boost disabled. We generate 1024 samples (with a random arbitrary center per sample) for 1000 times and measure the consumed CPU cycles. Then we convert the CPU cycles to the average number of samples per second for the comparison purpose with previous works.

The benchmark results of our scheme are shown in Table 1 (in the format of mean $\pm$ standard deviation). We also summarise the performance of previous works in Table 2. Since previous works [MW17, MR18, DWZ19] measured the number of generated samples per second running on CPUs with different frequencies, we scale all the numbers to be based on 4.2GHz[3]. In addition, since some previous works [MW17, MR18] require pre-computations to implement the sampling schemes, we summarise the pre-computation memory storage consumptions in Table 2. Because the TwinCDT method [MR18] provided different tradeoffs between the running speed and the pre-computation storage consumption, we show all 3 different sampling speeds and the corresponding pre-computation storage consumptions for each σ from [MR18]. Note that although our sampling scheme does not require pre-computations, however, the $\exp(x)$ implementation typically consumes a small

[2] Our implementation is available at https://github.com/raykzhao/gaussian_ac

[3] Since the authors of [MW17] did not provide the CPU frequency, we scale the benchmark results (online+offline) of the variant implemented by [DWZ19] instead.

Table 1: Number of Samples per Second for Our Scheme with Fixed σ at 4.2GHz (with $\lambda = 128$).

σ	Ref. ($\times 10^6$)	Constant-time ($\times 10^6$)
2	10.33 ± 0.18	8.96 ± 0.16
4	11.57 ± 0.18	10.87 ± 0.15
8	11.95 ± 0.17	11.61 ± 0.13
16	12.14 ± 0.16	12.00 ± 0.12
32	12.19 ± 0.15	12.21 ± 0.11
2^{15}	11.70 ± 0.13	11.57 ± 0.09

Table 2: Summary of Previous Works for Fixed σ at 4.2GHz (with $\lambda = 128$).

σ	Num. of samples ($\times 10^6$/sec)	Pre-computation storage (KB)
2 [MR18]	51.01/62.45/76.43	1.4/4.6/46
4 [MR18]	45.50/56.44/69.09	1.9/6.3/63
8 [MR18]	37.70/53.31/63.51	3/10/100
16 [MR18]	31.29/37.63/52.29	5.2/17/172
32 [MR18]	34.38/39.76/42.60	9.5/32/318
2^{15} [MW17]	1.78	$2^{5.4}$
4–2^{20} [DWZ19]	≈ 16.3	–[4]

amount of memory to store the coefficients of the polynomial approximation. For example, the polynomial approximation of the $\exp(x)$ in our constant-time implementation (adapted from [ZSS18]) has degree 10 with double precision coefficients, and therefore it consumes $(10 + 1) \cdot 8 = 88$ bytes.

From Table 1, our scheme has good performance for both small and large σ (11.65×10^6 samples per second for the non-constant time reference implementation and 11.20×10^6 samples per second for the constant-time implementation on average). In particular, our scheme has better performance for large σ since the number of trials becomes lower by Theorem 4 (note that the performance for $\sigma = 2^{15}$ is slightly slower than $\sigma = 32$ in Table 1 due to the larger l used by the rejection comparison steps, i.e. more randomness is required). The overhead introduced by the constant-time implementation is at most 13.33% in our benchmarks.

For $\sigma \in [2, 32]$, although the TwinCDT method [MR18] is 2.5x–7.3x faster than our reference implementation, however, this method requires a pre-computation with at least 1.4 KB memory consumption to store the CDT, while our scheme only requires at most several hundred bytes if considering all the polynomial approximation coefficients (including those functions used by the Box-Muller continuous Gaussian sampler). When scaling up σ, the TwinCDT method [MR18] also costs much larger amount of memory (the pre-computation storage size increases by a factor of 6.7–6.9 when σ changes from 2 to 32), and the performance becomes significantly worse (the number of samples per second decreases by 32.6–44.3% when σ changes from 2 to 32). In contrary, the pre-computation storage of our scheme is independent of σ and only relies on the precision requirements. Our scheme is also scalable and maintains good performance even for large $\sigma = 2^{15}$. In addition, for applications sampling from various σ such as [DLP14], one sampler subroutine implemented by using our scheme is able to serve all σ since the implementation does not require any pre-computations depending on σ, while the TwinCDT method [MR18] needs to pre-compute a different CDT for each σ.

[4] The base sampler and the Bernoulli sampler may require pre-computations depending on the implementation techniques.

Compared with [MW17] (online+offline) for $\sigma = 2^{15}$, our constant-time implementation[5] achieves better performance in terms of both timing (6.5x faster) and pre-computation storage (the implementation in [MW17] requires about 42 KB to implement the Knuth-Yao [KY76] offline batch sampler). Although our reference implementation is about 28.5% slower than [DWZ19], on the other hand, our scheme does not rely on any discrete Gaussian sampler implementations and the constant-time implementation perspective of [DWZ19] is unclear.

6 Conclusion

In conclusion, we generalise the idea from [Dev86] and present a compact and scalable arbitrary-centered discrete Gaussian sampling scheme over integers. Our scheme performs rejection sampling on rounded samples from a continuous normal distribution, which does not rely on any discrete Gaussian sampling implementations. We show that our scheme maintains good performance for $\sigma \in \left[2, 2^{15}\right]$ and needs no pre-computations related to any specific σ, which is suitable to implement applications that requires sampling from multiple different σ. In addition, we provide concrete rejection rate and error analysis of our scheme.

The performance of our scheme heavily relies on the underlying continuous Gaussian sampling algorithm. However, the Box-Muller sampler [ZCHW17, HLS18] employed in our implementation does not have the fastest sampling speed compared to other algorithms according to a survey [TLLV07]. The main reason behind the choice of the continuous Gaussian sampler in our implementation is because the Box-Muller sampler is very simple to implement in constant-time [HLS18]. If the side-channel perspective is not a concern, one may employ other more efficient non-constant time algorithms from the survey [TLLV07] to achieve a faster implementation of our scheme.

References

[BLL+15] Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance. In *ASIACRYPT (1)*, volume 9452 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2015.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2013.

[Dev86] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, NY, USA, 1986.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2014.

[DN12] Léo Ducas and Phong Q. Nguyen. Faster gaussian lattice sampling using lazy floating-point arithmetic. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2012.

[DWZ19] Yusong Du, Baodian Wei, and Huang Zhang. A rejection sampling algorithm for off-centered discrete gaussian distributions over the integers. *SCIENCE CHINA Information Sciences*, 62(3):39103:1–39103:3, 2019.

[5] Here we compare the performance of constant-time implementation because the implementation in [MW17] is constant-time.

[Fog] Agner Fog. VCL C++ vector class library. https://www.agner.org/optimize/vectorclass.pdf. Accessed: 2019-08-01.

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

[HLS18] Andreas Hülsing, Tanja Lange, and Kit Smeets. Rounded gaussians - fast and secure constant-time sampling for lattice-based crypto. In *Public Key Cryptography (2)*, volume 10770 of *Lecture Notes in Computer Science*, pages 728–757. Springer, 2018.

[Kar16] Charles F. F. Karney. Sampling exactly from the normal distribution. *ACM Trans. Math. Softw.*, 42(1):3:1–3:14, 2016.

[KY76] D. Knuth and A. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

[MAR17] Carlos Aguilar Melchor, Martin R. Albrecht, and Thomas Ricosset. Sampling from arbitrary centered discrete gaussians for lattice-based cryptography. In *ACNS*, volume 10355 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2017.

[MR18] Carlos Aguilar Melchor and Thomas Ricosset. CDT-based gaussian sampling: From multi to double precision. *IEEE Trans. Computers*, 67(11):1610–1621, 2018.

[MW17] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In *CRYPTO (2)*, volume 10402 of *Lecture Notes in Computer Science*, pages 455–485. Springer, 2017.

[Pei10] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2010.

[PFH+17] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. https://falcon-sign.info/, 2017. Accessed: 2018-10-31.

[Pre17] Thomas Prest. Sharper bounds in lattice-based cryptography using the rényi divergence. In *ASIACRYPT (1)*, volume 10624 of *Lecture Notes in Computer Science*, pages 347–374. Springer, 2017.

[PRR19] Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Simple, fast and constant-time gaussian sampling over the integers for falcon. Second PQC Standardization Conference, https://csrc.nist.gov/CSRC/media/Events/Second-PQC-Standardization-Conference/documents/accepted-papers/rossi-simple-fast-constant.pdf, 2019. Accessed: 2019-08-13.

[TLLV07] David B. Thomas, Wayne Luk, Philip Heng Wai Leong, and John D. Villasenor. Gaussian random number generators. *ACM Comput. Surv.*, 39(4):11, 2007.

[von51] John von Neumann. Various techniques used in connection with random digits. In A.S. Householder, G.E. Forsythe, and H.H. Germond, editors, *Monte Carlo Method*, pages 36–38. National Bureau of Standards Applied Mathematics Series, 12, Washington, D.C.: U.S. Government Printing Office, 1951.

[ZCHW17] Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, and William Whyte. NIST PQ submission: pqNTRUSign a modular lattice signature scheme. https://www.onboardsecurity.com/nist-post-quantum-crypto-submission, 2017. Accessed: 2019-08-01.

[ZSS18] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. FACCT: fast, compact, and constant-time discrete gaussian sampler over integers. *IACR Cryptology ePrint Archive*, 2018:1234, 2018.