Generic, Efficient and Isochronous Gaussian Sampling over the Integers

Shuo Sun[1,2], Yongbin Zhou[1,2,3], Yunfeng Ji[1,2], Rui Zhang[1,2], and Yang Tao[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China,
[3] School of Cyber Security, Nanjing University of Science and Technology, Nanjing, China
{sunshuo, zhouyongbin, jiyunfeng, r-zhang, taoyang}@iie.ac.cn

Abstract. Gaussian sampling over the integers is one of the fundamental building blocks of lattice-based cryptography. Among the extensively used trapdoor sampling algorithms, it's ineluctable until now. Under the influence of numerous side-channel attacks, it's still challenging to construct a Gaussian sampler that is generic, efficient, and resistant to timing attacks. In this paper, our contribution is three-fold.
First, we propose a secure, efficient exponential Bernoulli sampling algorithm. It can be applied to Gaussian samplers based on rejection samplings. We apply it to FALCON, a candidate of round 3 of the NIST post-quantum cryptography standardization project, and reduce its signature generation time by 13%-14%.
Second, we develop an isochronous Gaussian sampler based on rejection sampling. Our Algorithm can securely sample from Gaussian distributions with different standard deviations and arbitrary centers. We apply it to PALISADE (S&P 2018), an open-source lattice cryptography library. During the online phase of trapdoor sampling, the running time of the G-lattice sampling algorithm is reduced by 44.12% while resisting timing attacks.
Third, we improve the efficiency of the COSAC sampler (PQC 2020). The new COSAC sampler is 1.46x-1.63x faster than the original and has the lowest expected number of trials among all Gaussian samplers based on rejection samplings. But it needs a more efficient algorithm sampling from the normal distribution to improve its performance.

Keywords: Lattice-based cryptography · Gaussian sampler · Rejection sampling · Timing attacks · Trapdoor.

1 Introduction

Lattice-based cryptography has gained much attention due to its many attractive features. It allows us to build various powerful cryptographic primitives, such as fully homomorphic encryption [19], and has conjectured security against quantum computers [42]. Most lattice-based cryptographic schemes are based on

two main average-case problems, the short integer solution (SIS) problem [3], the learning with errors (LWE) problem [40], and their analogs over rings [31,29]. Discrete Gaussian distributions are at the core of security reduction proofs from the worst-case lattice problems to the average-case problems [33,7]. Therefore, they have great significance to the theoretical security of lattice-based cryptographic schemes.

In practice, it's notoriously difficult to sample from discrete Gaussian distributions effectively and securely, as demonstrated by numerous side-channel attacks [21,15,37,6] against the Gaussian sampler in the BLISS signature [11]. For this reason, some schemes replace Gaussian distributions with other distributions [28,41], like uniform distributions or binomial distributions, even if this ordinarily leads to performance and security degradation.

However, discrete Gaussian distributions are ineluctable in some situations. One prominent example is trapdoor sampling [20,36,32]. Trapdoor sampling can be used to construct many powerful cryptographic applications, including signature [20,39], (hierarchical) identity-based encryption ((H)IBE) [20,1,12], and attribute-based encryption (ABE) [43], etc. There are two extensively used trapdoor sampling algorithms at present. The first one is proposed by Gentry, Peikert, and Vaikuntanathan [20] (GPV trapdoor sampler), while the second one is proposed by Micciancio and Peikert [32] (MP trapdoor sampler). Both of them require to sample from the Gaussian distributions over the integers with different standard deviations and arbitrary centers [12,39,18,24]. To resist timing attacks, the former requires the Gaussian sampler over the integers doesn't leak the information of the standard deviation σ, center c, and output z [17], while the latter requires the sampler doesn't leak the information of c and z. In practical implementations, the trapdoor samplers usually employ a generic Gaussian sampler over the integers whose precomputation cost doesn't vary with different σ and c. So it's important to propose a Gaussian sampler over the integers which is generic, efficient, and resistant to timing attacks.

There are two strategies to design generic Gaussian samplers over the integers. One is based on the convolution theorems of discrete Gaussian distributions, and another is based on rejection samplings. The convolution sampler [34] belongs to the former. It's easy to make its running time independent of its inputs σ, c and output z. However, its efficiency highly relies on online/offline skills, and its whole running time is not competitive. For Gaussian samplers based on rejection samplings, the rejection sampling algorithm seems to be inherently costly to turn into a constant-time algorithm. FALCON's sampler [23] is not constant-time, but isochronous concerning its inputs σ, c, and output z. Isochrony is sufficient to resist timing attacks. But FALCON's sampler is only suitable for small σ. Karney's sampler [27] and the COSAC sampler [44] are suitable for large σ. However, Karney's sampler doesn't consider how to resist timing attacks, while the COSAC sampler may reveal its input σ. One crucial step of Gaussian samplers based on rejection samplings is sampling an exponential Bernoulli variable $\mathcal{B}_p$ with probability $p = \exp(-x)$ $(x \geq 0)$ being true. It's

a time-consuming module, and the mainstream solution is approximating the exponential function by a polynomial [45,5].

In some lattice cryptography libraries, such as PALISADE [8,9,22], it's one of the fundamental building blocks to sample from the Gaussian distributions over the integers. To implement MP trapdoor sampler, the lattice cryptography library PALISADE [8] employs the convolution sampler and Karney's sampler as the Gaussian samplers over the integers. The convolution sampler is only used in the offline phase, as it doesn't seem easy to get an efficient implementation of the convolution sampler. PALISADE employs Karney's sampler to obtain a more efficient implementation, although Karney's sampler can't resist timing attacks. Thus, it deserves more effort to design a Gaussian sampler over the integers which is generic, efficient, and resistant to timing attacks.

1.1 Our Contribution

In this paper, we mainly focus on Gaussian samplers over the integers based on rejection samplings. We first propose an exponential Bernoulli sampling algorithm sampling the exponential Bernoulli variables. It's a basic tool to construct the Gaussian samplers based on rejection samplings. Then we utilize the discrete Gaussian distribution with a small standard deviation as the base distribution and design an isochronous Gaussian sampler based on rejection sampling. The COSAC sampler is a rejection sampling algorithm using the normal distribution as the base distribution. We reduce its expected number of trials to improve the efficiency. Therefore our contribution is three-fold.

The inspiration of our exponential Bernoulli sampling algorithm comes from von Neumann's algorithm that samples from the exponential distribution. As far as we know, our work is the first one that securely applies this idea to the discrete Gaussian sampling to improve the efficiency of the cryptographic scheme. Compared with the mainstream method approximating the exponential function by a polynomial [45], our algorithm has the following advantages:

- Our algorithm has higher efficiency while resist timing attacks in rejection sampling scenes. On the one hand, we improve the performance by reducing floating-point operations significantly. On the other hand, when our algorithm is applied to the rejection sampling algorithm resistant to timing attacks, its output only relies on the public rejection rate and its running time is independent of its input, therefore our algorithm can resist timing attacks.
- We apply our algorithm to FALCON [39], a lattice-based signature scheme of the third-round candidates in the NIST post-quantum cryptography standardization project (NIST PQ project). When the LDL tree[4] has been built upon the secret key, the signature generation time of the new implementation decreases by $13\% - 14\%$.

[4] The LDL tree is a data structure to reduce the complexity of the Gaussian sampling over rings during the signature generation. It depends only on the secret key.

Table 1: Comparison between our work and existing generic[‡] samplers at 3.6 GHz

Algorithm	Isochrony[*]	σ	Memory (KB)	Number of Samples ($\times 10^6$/sec)
Convolution Sampler [34]	Type I	$3 - 2^{15}$	$O(1)$ ($\approx 2^{5.4}$)	1.53
Karney's Sampler [27]	$\times$	$1 - 2^{20}$	$O(1)$ (< 1)	8.10
DWZ Sampler [10]	$\times$	$4 - 2^{20}$	$O(1)$ (< 1)	13.97
FALCON's Sampler [23]	Type I	$1.29 - 1.82$	$O(\sigma_{\max})$ (< 1)	7.53
COSAC Sampler (AVX2) [44]	Type III[⋆]	$2 - 2^{20}$	$O(1)$ (< 1)	6.81
Our work Algorithm 4[†]	Type I	$2 - 2^{20}$	$O(1)$ (< 1)	7.12
	Type II	$2 - 2^{20}$	$O(1)$ (< 1)	13.55
Our work Algorithm 8 (AVX2)	Type III[⋆]	$2 - 2^{20}$	$O(1)$ (< 1)	11.07

[‡] "Generic" means that the precomputation cost doesn't vary with different σ and c. The precomputation of FALCON's Sampler is determined by the maximum standard deviation $\sigma_{\max}$.

[*] In this column, "$\times$" means the sampler can't resist timing attacks; "Type I" means the sampler is isochronous concerning σ, c and z; "Type II" means the sampler is isochronous concerning c and z; "Type III" means the sampler is only isochronous concerning z.

[†] In section 4.2, we provide two base samplers. Here we utilize the CDT sampler as the base sampler.

[⋆] The reference implementation of the COSAC sampler batches the random number generation to hide its output z, but its rejection rate still seems to depend on c. Our implementation of Algorithm 8 may have the same problem.

We derive the idea of our isochronous Gaussian sampler from Karney's sampler [27] and FALCON's sampler [23]. We employ the rejection sampling strategy[5] of Karney's sampler to design the sampler, then apply a similar technique as FALCON's sampler to make it isochronous concerning its inputs (standard deviation σ and center c) and output (sampling value z). There are four differences between the techniques in our work and [23]: (1) We use Algorithm 3 instead of the polynomial approximation to sample the exponential Bernoulli variable; (2) We provide two base samplers the cumulative distribution table (CDT) sampler and the binary sampler (Algorithm 5); (3) To make the rejection rate independent of σ, the parameter $C(\sigma)$ in Algorithm 4 relies on the ratio σ/σ_0 rather than σ (Lemma 8); (4) We need Algorithm 6 to sample the integers from $[0, \lceil k \rceil)$ uniformly. Our discrete Gaussian sampling algorithm has the following advantages:

- As shown in Table 1, Algorithm 4 has the best performance while the samplers consider timing attacks.

[5] The rejection sampling strategy is the way of sampling the discrete Gaussian distributions with different σ and arbitrary c by the base sampler.

- Compared with FALCON's sampler, the memory consumption and running time of Algorithm 4 are almost unaffected by the maximum standard deviation $\sigma_{\max}$, so it's suitable for large σ. When we use the parameter $C(\sigma)$ in Algorithm 4 to hide σ, we could adjust $C(\sigma)$ to the minimum σ required by the cryptographic scheme, such that our sampler has the better performance (see section 4.3 for a detailed discussion).
- We replace Karney's sampler in PALISADE with our Gaussian sampler to speed up the online phase of the MP trapdoor sampler. The running time of the G-lattice sampling algorithm in the online phase decreases by 44.12%.

The core of our new COSAC sampler is to adjust the rejection sampling strategy to the center. The expected number of trials of the new COSAC sampler is half of that of the original. As the standard deviation increases, its expected number of trials converges to 1, which is the best in theory for the rejection sampling algorithms. However, the new COSAC sampler is only 1.46x-1.63x faster than the original due to the additional operations to hide the output. As shown in Table 1, it is not faster than Algorithm 4 and needs a more efficient algorithm sampling from the normal distribution to improve its performance.

1.2 Related Works

Some works [16,2,27] have employed the idea of von Neumann's algorithm to elegantly sample from a continuous probability distribution with the probability density function proportional to $\exp(-G(x))$, where $G(x)$ is some simple function. The normal distribution is one of the continuous probability distributions. However, it's hard to make the algorithms isochronous. To the best of our knowledge, there is no work to employ the idea to improve the efficiency of cryptographic schemes in practice.

Du et al. [10] proposed a Gaussian sampler over the integers (DWZ sampler) utilizing the rejection sampling strategy of Karney's sampler [27]. It's similar to Algorithm 4. However, they chose the insecure binary sampler as the base sampler and computed the exponential function directly. Therefore, its side-channel resistance perspective is unclear. Although it's faster than Karney's sampler, it doesn't solve the crucial security problem of Karney's Sampler. In addition, we employ the CDT sampler as the base sampler of Algorithm 4 and get better performance. There are some not generic, but efficient Gaussian samplers. The CDT sampler and Knuth-Yao sampler are often used as the base samplers. Karmakar et al. [26] crafted a constant-time and efficient implementation of the Knuth-Yao sampler. The Twin-CDT sampler [30] can sample from the discrete Gaussian distribution with the fixed σ and arbitrary c. It's constant-time.

2 Preliminaries

2.1 Notations

Let $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{N}$ be the set of real numbers, integers and non-negative integers respectively. For a distribution D, we use the notation $x \xleftarrow{\$} D$ to mean that x is

chosen according to the distribution D. If S is a set, then $x \xleftarrow{\$} S$ means that x is sampled uniformly at random from S. The left arrow $\leftarrow$ denotes the assignment operator. We use the notation $\mathcal{B}_p$ to denote the Bernoulli distribution with parameter p. A random variable with $\mathcal{B}_p$ takes the true value with probability p and the false value with probability $1-p$. We denote the logarithm with base 2 by log and the one with base e by ln.

2.2 Gaussian Distributions

For any $\sigma, c \in \mathbb{R}$ with $\sigma > 0$, the Gaussian function with parameters σ and c over $\mathbb{R}$ is defined as $\rho_{\sigma,c}(x) = \exp(-\frac{(x-c)^2}{2\sigma^2})$. We denote the normal (or continuous Gaussian) distribution with parameters σ and c over $\mathbb{R}$ by $\mathcal{N}(c, \sigma^2)$, which has the probability density function $\rho_{\sigma,c}(x)/(\sigma\sqrt{2\pi})$. For any countable set $S \subsetneq \mathbb{R}$, we denote $\rho_{\sigma,c}(S)$ by the sum $\sum_{x\in S} \rho_{\sigma,c}(x)$. If $\rho_{\sigma,c}(S)$ is finite, we define the discrete Gaussian distribution with parameters σ and c over S as $D_{S,\sigma,c}(x) = \rho_{\sigma,c}(x)/\rho_{\sigma,c}(S)$, and denote the distribution by $D_{S,\sigma,c}$. The parameter σ (resp. c) is often called the standard deviation (resp. center) of the distribution. Note that when $c = 0$, we omit it in index notation, e.g. $\rho_\sigma(x) = \rho_{\sigma,0}(x)$ and $D_{S,\sigma}(x) = D_{S,\sigma,0}(x)$.

2.3 Smoothing Parameter

The smoothing parameter is a lattice quantity proposed by Micciancio and Regev [33]. For $\epsilon > 0$, the smoothing parameter $\eta_\epsilon(\Lambda)$ of a lattice Λ is the smallest value $\sigma > 0$ such that $\rho_{\frac{1}{\sigma\sqrt{2\pi}}}(\Lambda^* \backslash \{0\}) \leq \epsilon$, where Λ^* denotes the dual of Λ. The initial definition [33] of the smoothing parameter is $\sqrt{2\pi}$ times that our definition, as it is defined by the Gaussian parameter $s = \sigma\sqrt{2\pi}$.

Lemma 1 ([33]). *For any positive real $\epsilon > 0$, we have $\eta_\epsilon(\mathbb{Z}) \leq \eta^+_\epsilon(\mathbb{Z})$:*

$$\eta^+_\epsilon(\mathbb{Z}) = \frac{1}{\pi}\sqrt{\frac{1}{2}\ln\left(2 + \frac{2}{\epsilon}\right)}.$$

The following lemma states that the total Gaussian measure over $\mathbb{Z}$, i.e. $\rho_{\sigma,c}(\mathbb{Z})$, is essentially the same for any center c when the standard deviation σ exceeds the smoothing parameter $\eta_\epsilon(\mathbb{Z})$.

Lemma 2 ([33,20]). *For any $\epsilon \in (0,1)$, $\sigma > \eta_\epsilon(\mathbb{Z})$ and $c \in \mathbb{R}$, we have*

$$\rho_{\sigma,c}(\mathbb{Z}) \in \left[\frac{1-\epsilon}{1+\epsilon}, 1\right] \cdot \rho_\sigma(\mathbb{Z}).$$

We need the following lemma to make the running time of a discrete Gaussian sampling algorithm independent of σ. One may get a similar result by the proof of Lemma 4.4 in [33]. The difference here is that σ is larger than $\eta^+_\epsilon(\mathbb{Z})$ rather than $\eta_\epsilon(\mathbb{Z})$. We give a brief proof in Appendix A.

Lemma 3. *For any $\epsilon > 0$, $\sigma > \eta^+_\epsilon(\mathbb{Z})$, we have:*

$$\rho_\sigma(\mathbb{Z}) \in [1, 1+\epsilon] \cdot \sigma\sqrt{2\pi}.$$

2.4 Rényi Divergence

We recall the definition of the Rényi divergence, which is extensively used in the cryptographic security proofs currently.

Definition 1 ([4]). *Let $\mathcal{P}$, $\mathcal{Q}$ be two distributions such that Supp($\mathcal{P}$) $\subseteq$ Supp($\mathcal{Q}$). For $a \in (1, +\infty)$, we define the Rényi divergence of order a by*

$$R_a(\mathcal{P}, \mathcal{Q}) = \left(\sum_{x \in Supp(\mathcal{P})} \frac{\mathcal{P}(x)^a}{\mathcal{Q}(x)^{a-1}} \right)^{\frac{1}{a-1}}.$$

In addition, we define the Rényi divergence of $+\infty$ by $R_\infty(\mathcal{P}, \mathcal{Q}) = \max\limits_{x \in Supp(\mathcal{P})} \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}$.

The Rényi divergence is not a distance, but it does verify cryptographically useful properties. We recall some important properties of the Rényi divergence from [4,14].

Lemma 4 ([4]). *Let $a \in (1, +\infty]$. Let $\mathcal{P}$ and $\mathcal{Q}$ denote distributions with Supp($\mathcal{P}$) $\subseteq$ Supp($\mathcal{Q}$). Then the following properties hold:*

- ***Data Processing Inequality.*** *For any function f, where $f(\mathcal{P})$ (resp. $f(\mathcal{Q})$) denotes the distribution of $f(y)$ induced by sampling y from $\mathcal{P}$ (resp. $\mathcal{Q}$), we have:*
$$R_a(f(\mathcal{P}), f(\mathcal{Q})) \leq R_a(\mathcal{P}, \mathcal{Q}).$$
- ***Multiplicativity.*** *For two families of distributions $(\mathcal{P}_i)_i, (\mathcal{Q}_i)_i$,*
$$R_a(\prod\nolimits_i \mathcal{P}_i, \prod\nolimits_i \mathcal{Q}_i) = \prod\nolimits_i R_a(\mathcal{P}_i, \mathcal{Q}_i).$$
- ***Probability Preservation.*** *For any event $E \subseteq Supp(Q)$ and $a \in (1, +\infty)$,*
$$\mathcal{Q}(E) \geq \mathcal{P}(E)^{\frac{a}{a-1}} / R_a(\mathcal{P}, \mathcal{Q}) \quad and \quad \mathcal{Q}(E) \geq \mathcal{P}(E) / R_\infty(\mathcal{P}, \mathcal{Q}).$$

In practice, when we approximate the original distribution $D_{\mathbb{Z},\sigma,c}$ by sampling from a finite set, we can use the following lemma to bound the Rényi divergence between the real and ideal distributions.

Lemma 5 ([38]). *Let $\mathcal{P}$ and $\mathcal{Q}$ be two distributions such that Supp($\mathcal{P}$) $\subseteq$ Supp($\mathcal{Q}$). For any $x \in Supp(\mathcal{P})$, if there exists $\delta > 0$ such that $\frac{\mathcal{P}(x)}{\mathcal{Q}(x)} \leq 1 + \delta$ then, for $a \in (1, +\infty)$:*

$$R_a(\mathcal{P}, \mathcal{Q}) \leq 1 + \delta.$$

When we use the double type or 64-bit integers to approximate floating-point numbers, which results in the relative error between the real and ideal distributions, the following lemma gives a bound of the Rényi divergence between the distributions.

Lemma 6 ([38]). *Let $\mathcal{P}$ and $\mathcal{Q}$ be two distributions of the same support S. Suppose that the relative error between $\mathcal{P}$ and $\mathcal{Q}$ is bounded: $\forall x \in S$, $\exists \delta > 0$, such that $\left| \frac{\mathcal{P}(x)}{\mathcal{Q}(x)} - 1 \right| \leq \delta$. Then, for $a \in (1, +\infty)$:*

$$R_a(\mathcal{P}, \mathcal{Q}) \leq \left(1 + \frac{a(a-1)\delta^2}{2(1-\delta)^{a+1}}\right)^{\frac{1}{a-1}} \underset{\delta \to 0}{\sim} 1 + \frac{a\delta^2}{2}.$$

2.5 Isochronous Algorithm

When an algorithm is applied to different scenarios, the sensitive variables of the algorithm may be different. To resist against timing attacks, we only need to keep the running time of the algorithm from leaking sensitive parameters for a particular scenario. In this work, when we show that our algorithms are provably resistant against timing attacks, we use the following definition of the isochronous algorithm to capture this idea.

Definition 2 ([23]). *Let $\mathcal{A}$ be a (probabilistic or deterministic) algorithm with set of input variables $\mathcal{I}$, set of output variables $\mathcal{O}$, and let $\mathcal{S} \subseteq \mathcal{I} \cup \mathcal{O}$ be the set of sensitive variables. We say that A is perfectly isochronous with respect to $\mathcal{S}$ if its running time is independent of any variables in $\mathcal{S}$.*

In addition, we say that $\mathcal{A}$ is statistically isochronous with respect to $\mathcal{S}$ if there exists a distribution $\mathcal{D}$ independent of all the variables in $\mathcal{S}$, such that the running time of $\mathcal{A}$ is statistically close (for a clearly identified divergence) to $\mathcal{D}$.

In this paper, we refer to the implementations of FALCON and previous Gaussian samplers [23,39] and assume that the four basic operators over integers, the addition, subtraction, and multiplication over floating-point numbers, the rounding, ceiling, floor, and bitwise operations are isochronous. For the lattice-based trapdoor sampling, the short lattice basis determines σ of the Gaussian distribution over the integers, while c is usually variable. As the floating-point division rarely offers constant-time execution guarantee [45], we precompute the parameters $1/\sigma$, k, $\lceil k \rceil$ and P_k in section 4. We can also precompute $S = \rho_{\sigma,c}(\mathbb{Z})$ in section 5 by Lemma 2[6]. While the implementation of FALCON's sampler only precomputes $1/\sigma$[7], Algorithm 4 may have to precompute a few more parameters. But its memory consumption is also tolerable (see Table 1 and section 4.3).

2.6 Von Neumann's Algorithm

Von Neumann's algorithm can sample a variable from the exponential distribution and avoid floating-point operations. It's the foundation of our exponential Bernoulli sampling algorithm. We review it in Algorithm 1.

Lemma 7. *The variable x output by Algorithm 1 obeys the exponential distribution.*

Proof. We first analyze the probability that n is odd in step 7 of Algorithm 1 for any $x_2 \in [0, 1)$. If $n = n_0$ in step 7, there are $n_0 + 1$ variables $u_1, u_2, ..., u_{n_0+1}$ uniformly sampled from [0, 1) in step 2 and step 5. These variables satisfy that $x_2 > u_1 > ... > u_{n_0}$ and $u_{n_0} \leq u_{n_0+1}$. The probability that $u_1, ..., u_{n_0}$ are all less than x_2 is $x_2^{n_0}$. In addition, the probability that they are in descending order,

[6] If the adversary makes no more than 2^{64} queries, for a 256-bit cryptographic scheme, $\epsilon = 2^{40}$ and $\sigma \geq 2 > \eta_\epsilon^+(\mathbb{Z})$ can ensure security through the analysis in section 4.2.

[7] Assuming the floating-point division is isochronous [23], FALCON's sampler only requires to precompute the cumulative distribution table.

Algorithm 1: Von Neumann's Algorithm

Output: A variable x from the exponential distribution

1: $x_1 \leftarrow 0$, $x_2 \xleftarrow{\$} [0,1)$, $n \leftarrow 0$
2: $t_1 \leftarrow x_2$, $t_2 \xleftarrow{\$} [0,1)$
3: **while** $t_1 > t_2$ **do**
4: $\quad n \leftarrow n+1$
5: $\quad t_1 \leftarrow t_2$, $t_2 \xleftarrow{\$} [0,1)$
6: **end while**
7: **if** n *is odd* **then**
8: $\quad x_1 \leftarrow x_1 + 1$
9: $\quad$ *goto step* 2
10: **end if**
11: $x \leftarrow x_1 + x_2$
12: **return** x

which is one of the possible $n_0!$ permutations, is $x_2^{n_0}/n_0!$. As $u_{n_0} \leq u_{n_0+1}$, the probability that $n = n_0$ in step 7 is $x_2^{n_0}/n_0! - x_2^{n_0+1}/(n_0+1)!$. For any $x_2 \in [0,1)$, the probability that n is even in step 7 is

$$1 - x_2 + \frac{x_2^2}{2!} - \frac{x_2^3}{3!} + ... = \exp(-x_2).$$

Because x_2 is sampled from $[0,1)$ uniformly, the probability that n is odd averaged over x_2 is $\int_0^1 (1 - \exp(-x_2))dx_2 = \exp(-1)$. Thus, the probability density that the algorithm terminates with a particular value of x_1 and x_2 is $\exp(-(x_1 + x_2))$, as required. □

3 A Tool: Exponential Bernoulli Sampling Algorithm

In this section, we propose a tool, the exponential Bernoulli sampling algorithm, to construct the Gaussian samplers based on rejection samplings. We first introduce the basic algorithm and analyze its relative error. Then we show how to make it isochronous in the particular application. Finally, we apply it to the FALCON signature.

3.1 Basic Exponential Bernoulli Sampling Algorithm

When sampling a Bernoulli variable with probability $\exp(-x)$ being true, the polynomial approximation method needs 14 floating-point multiplications in which 12 floating-point multiplications are used to compute the exponential function [45,39]. To avoid these 12 floating-point multiplications, we can sample the Bernoulli variable utilizing step 2 - step 6 in Algorithm 1 for any $x \in [0,1)$. For any $x \geq 0$, let $x = u_1 \ln 2 + u_2$ where u_1 is a non-negative integer and $0 \leq u_2 < \ln 2$, then $\exp(-x)$ can be written as $\exp(-x) = 2^{-u_1} \exp(-u_2)$. The

first part 2^{-u_1} can be sampled efficiently in binary form, and the second part $\exp(-u_2)$ can be sampled efficiently too as $u_2 \in [0,1)$. We present the basic exponential Bernoulli sampling algorithm in Algorithm 2.

Algorithm 2: Basic Exponential Bernoulli Sampling Algorithm

Input: $x \geq 0$

Output: $b \xleftarrow{\$} \mathcal{B}_{\exp(-x)}$

1: $u_1 \leftarrow \lfloor x/\ln 2 \rfloor$, $u_2 \leftarrow x - u_1 \cdot \ln 2$
2: $r_1 \xleftarrow{\$} \{0, 1, ..., 2^{u_1} - 1\}$
3: $b_1 \leftarrow (r_1 = 0)$
4: $n \leftarrow 0$
5: $r_2 \leftarrow u_2$, $r_3 \xleftarrow{\$} [0,1)$
6: **while** $r_2 > r_3$ **do**
7: $\quad n \leftarrow n + 1$
8: $\quad r_2 \leftarrow r_3$, $r_3 \xleftarrow{\$} [0,1)$
9: **end while**
10: $b_2 \leftarrow (n \ is \ even)$
11: $b \leftarrow b_1 \ \& \ b_2$
12: **return** b

Theorem 1. *The output b of Algorithm 2 obeys the Bernoulli distribution with probability $\exp(-x)$ being true.*

Proof. The probability that the variable b_1 is true is 2^{-u_1}. Because step 5 - step 9 in Algorithm 2 is equivalent to step 2 - step 6 in Algorithm 1, the probability that the variable b_2 is true is $\exp(-u_2)$ according to the proof of Lemma 7. As $x = u_1 \ln 2 + u_2$, the probability that the output b is true is $\exp(-x)$. □

In the next theorem, we estimate the expected sampling number of r_3 which is closely related to the efficiency of Algorithm 2.

Theorem 2. *For any $u_2 \in [0, \ln 2)$, in one running process of Algorithm 2, the expected sampling number S of r_3 is less than 2.*

Proof. The probability that $n = n_0$ in step 10 of Algorithm 2 is $u_2^{n_0}/n_0! - u_2^{n_0+1}/(n_0+1)!$. If $n = n_0$ in step 10, the sampling number of r_3 is $n_0 + 1$. The expected sampling number S of r_3 can be written as

$$S = \sum_{n_0=0}^{\infty} \left(\frac{u_2^{n_0}}{n_0!} - \frac{u_2^{n_0+1}}{(n_0+1)!} \right) \cdot (n_0 + 1) = \exp(u_2) < 2,$$

the last inequality is deduced from $u_2 < \ln 2$. □

When we prove the correctness of Algorithm 2 in Theorem 1, we treat it as an ideal algorithm that samples the variables b_1 and b_2 with the exact parameters 2^{-u_1} and $\exp(-x + u_1 * \ln 2)$. In practice, it's necessary to bound the relative error of Algorithm 2 due to the approximations of u_2 and r_3.

Theorem 3. *Let $P^{(R)}$ be the probability that the output b of the real algorithm is true where the floating numbers u_2, r_2, r_3 of Algorithm 2 keep m ($m \geq 5$) significant bits. Let $P^{(I)}$ be the probability that the output b of the ideal algorithm is true where u_2, r_2, r_3 have infinite significant bits. Then the relative error between $P^{(R)}$ and $P^{(I)}$ satisfies that*

$$\frac{|P^{(R)} - P^{(I)}|}{P^{(I)}} \leq 2^{-m} + 2^{-2m} + 2^{-m+3}.$$

Proof. Let $P_1^{(R)}$ and $P_2^{(R)}$ be the probabilities that the variables b_1 and b_2 of the real algorithm are true respectively, $P_1^{(I)}$ and $P_2^{(I)}$ be the probabilities that the variables b_1 and b_2 of the ideal algorithm are true respectively. Because u_1 is an integer and the sampling process of b_1 (step 2 - step 3) only involves the integer operations, $P_1^{(R)}$ and $P_1^{(I)}$ should be the same exactly. Thus, the relative error between $P^{(R)}$ and $P^{(I)}$ is the same as that between $P_2^{(R)}$ and $P_2^{(I)}$

$$\frac{|P^{(R)} - P^{(I)}|}{P^{(I)}} = \frac{|P_1^{(R)} P_2^{(R)} - P_1^{(I)} P_2^{(I)}|}{P_1^{(I)} P_2^{(I)}} = \frac{|P_2^{(R)} - P_2^{(I)}|}{P_2^{(I)}}. \tag{1}$$

The difference between $P_2^{(R)}$ and $P_2^{(I)}$ stems from the approximations of u_2 and r_3. We define an intermediate algorithm where u_2 has m significant bits and r_2 and r_3 have infinite bits. Let $P_2^{(IN)}$ be the probability that the variable b_2 in the intermediate algorithm is true. Then we have

$$\frac{|P_2^{(R)} - P_2^{(I)}|}{P_2^{(I)}} \leq \frac{|P_2^{(R)} - P_2^{(IN)}|}{P_2^{(I)}} + \frac{|P_2^{(IN)} - P_2^{(I)}|}{P_2^{(I)}} \tag{2}$$

As u_2 retains m significant bits and $u_2 \in [0, \ln 2)$, the difference between the approximate number $u_2^{(R)}$ and the accurate number $u_2^{(I)}$ is not greater than 2^{-m}. So we have

$$\begin{aligned} \frac{|P_2^{(IN)} - P_2^{(I)}|}{P_2^{(I)}} &= \left| \frac{\exp(-u_2^{(R)}) - \exp(-u_2^{(I)})}{\exp(-u_2^{(I)})} \right| = \left| \exp(-(u_2^{(R)} - u_2^{(I)})) - 1 \right| \\ &\leq \left| u_2^{(R)} - u_2^{(I)} \right| + \left(u_2^{(R)} - u_2^{(I)} \right)^2 \leq 2^{-m} + 2^{-2m}. \end{aligned} \tag{3}$$

We will next analyze the process of the while loop (step 6 - step 9) of Algorithm 2. For the i-th while loop ($i \geq 1$), we denote A_i as the event $r_2 = r_3$, B_i as the event $r_2 > r_3$ and C_i as the event $r_2 < r_3$. In the intermediate algorithm, the uniformly random floating-point number $r_3^{(I)}$ has infinite significant bits, so event A_i will never happen for any $i \geq 1$. In the real algorithm, both $r_2^{(R)}$ and $r_3^{(R)}$ have m significant bits and $r_3^{(R)}$ is sampled uniformly, the probability of event A_i is exactly 2^{-m} for any $i \geq 1$. Our goal is to calculate the probability sum of all A_i in the real algorithm which is the upper bound of $|P_2^{(R)} - P_2^{(IN)}|$.

For $i \geq 2$, event B_{i-1} must have happened before the i-th loop, which means $\Pr\{B_i\} = \Pr\{B_{i-1}\} \cdot r_2^{(R)}$ and $\Pr\{A_i\} = \Pr\{B_{i-1}\} \cdot 2^{-m}$. Because $r_2^{(R)}$ is always no more than $u_2^{(R)}$ and $u_2^{(R)} \leq u_2^{(I)} + 2^{-m}$, we have $\Pr\{A_i\} \leq \left(u_2^{(I)} + 2^{-m}\right)^{i-1} \cdot 2^{-m}$ for any $i \geq 1$. As $u_2^{(I)} \in [0, \ln 2)$ and $m \geq 5$, we can give an upper bound:

$$\frac{|P_2^{(R)} - P_2^{(IN)}|}{P_2^{(I)}} \leq \frac{\sum_{i=1}^{\infty} \Pr\{A_i\}}{\exp\left(-u_2^{(I)}\right)} \leq 2^{-m+1} \cdot \sum_{i=0}^{\infty} \left(u_2^{(I)} + 2^{-m}\right)^i \leq 2^{-m+3}. \quad (4)$$

Combining the constraints (1) (2) (3) (4), we can bound the relative error between $P^{(R)}$ and $P^{(I)}$,

$$\frac{\left|P^{(R)} - P^{(I)}\right|}{P^{(I)}} \leq 2^{-m} + 2^{-2m} + 2^{-m+3},$$

which concludes the proof. □

In our implementation of Algorithm 2, we use the double type meeting the IEEE-754 standard in the floating-point arithmetics and 64-bit integers to approximate the floating numbers u_2, r_2, r_3, so the number m of significant bits is at least 52. The relative error of our implementation is no more than

$$2^{-52} + 2^{-104} + 2^{-49} \leq 2^{-48}.$$

3.2 Isochronous Exponential Bernoulli Sampling Algorithm

In this subsection, we focus on how to design an isochronous exponential Bernoulli sampling algorithm to resist timing attacks. It seems difficult to make the input and output of Algorithm 2 isochronous with respect to its running time while retaining its efficiency. Because the sampling number of r_3 determining the value of b_2 is closely related to the running time of Algorithm 2. To overcome the obstacle, we restrict the application of our algorithm to the rejection sampling algorithm. In these applications, the distribution of output b only relies on the rejection probability of the rejection sampling algorithm. For the rejection sampling algorithm resisting timing attacks, the rejection probability can't reveal the information of the sensitive variables. Therefore there is no need to hide the value of output b. In Algorithm 3, we successfully make the input independent of the running time while retaining the efficiency.

Theorem 4. *The output of Algorithm 3 obeys the Bernoulli distribution with probability* $\exp(-x)$ *being true.*

Proof. After step 3, r_1 is a uniformly random integer in $[0, 2^{u_1} - 1]$. The probability that b_1 is true is 2^{-u_1}. There are two cases where b_2 is true. The first one is $u_2 < r_3$ in step 7, and the second one is that $u_2 > r_3$ in step 7 and n is even in step 12. As $u_2 < t$, their probabilities are $1 - u_2$ and $\sum_{i=1}^{\infty} \left(u_2^{2i}/(2i)! - u_2^{2i+1}/(2i+1)!\right)$ respectively. We can get that the probability that b_2 is true is $\exp(-u_2)$ by adding the two probabilities, which concludes the proof. □

Algorithm 3: Isochronous Exponential Bernoulli Sampling Algorithm

Input: $0 \leq x \leq h \cdot \ln 2$, public parameter $t \in [\ln 2, 1]$

Output: $b \xleftarrow{\$} \mathcal{B}_{\exp(-x)}$

$u_1 \leftarrow \lfloor x/\ln 2 \rfloor$, $u_2 \leftarrow x - u_1 \cdot \ln 2$
$r_1 \xleftarrow{\$} \{0, 1, ..., 2^h - 1\}$
$r_1 \leftarrow r_1 \ \& \ (2^{u_1} - 1)$
$b_1 \leftarrow (r_1 = 0)$
$n \leftarrow 0$
$r_2 \leftarrow t$, $r_3 \xleftarrow{\$} [0, 1)$
$b_2 \leftarrow (u_2 < r_3)$
while $r_2 > r_3$ **do**
    $n \leftarrow n + 1$
    $r_2 \leftarrow r_3$, $r_3 \xleftarrow{\$} [0, 1)$
end while
$b_2 \leftarrow b_2 \mid (n \ is \ even)$
$b \leftarrow b_1 \ \& \ b_2$
return b

Theorem 5. *Algorithm 3 is perfectly isochronous with respect to its input x. What's more, in one running process of Algorithm 3, the expected sampling number S of r_3 is* $\exp(t)$.

Proof. We need to prove that in Algorithm 3, u_1 and u_2 are respectively independent of the running times of sampling b_1 and b_2. The sample of b_1 consists of one uniform sampling, one AND operation, and one comparison, which is independent of u_1. The sample of b_2 consists of the uniform samplings of r_3, the comparisons of r_2 and r_3, one comparison of u_2 and r_3. The sampling number of r_3 is the same as the number of comparisons between r_2 and r_3. The distribution of the sampling number of r_3 depends on t rather than u_2. The expected sampling number S of r_3 is

$$S = \sum_{n_0=0}^{\infty} \left(\frac{t^{n_0}}{n_0!} - \frac{t^{n_0+1}}{(n_0+1)!} \right) \cdot (n_0 + 1) = \exp(t).$$

□

In our implementation of Algorithm 3, we set $t = 178 * 2^{-8}$ to avoid the error of the floating-point number. So the relative error of Algorithm 3 stems from the approximations of u_2 and r_3, which is the same as Algorithm 2. With the almost same proof of Theorem 3, we can get the same upper bound of the relative error of Algorithm 3 in practice.

Laziness Technique. When we use 64-bit integers to represent r_1, r_2, and r_3, there are at most $64 \cdot (1 + e)$ bits randomness required on average during each run of Algorithm 3. These integers are only involved in the comparison operations, so we can employ the laziness technique [13,23] to speed up the implementation of

Algorithm 3. When at least one of the two compared 64-bit integers is sampled uniformly, the comparison can be determined by the first i significant bits, except with probability 2^{-i} (exactly when the first i bits of the two integers match). Therefore it is unnecessary to sample all bits of the integers at one time. In our implementation, the 64-bit integer is sampled 8 bits by 8 bits until the comparison is determined or all 64 bits are sampled. It is easy to estimate that each sampled integer needs less than 9 random bits on average in this way. So the expected number of random bits is less than $9 \cdot (1+e)$ in one running process of Algorithm 3 with this technique.

It should be noted that the random number r_3 in step 6 of Algorithm 3 has to be compared with t and u_2. We need to carefully select a value of the public parameter t to ensure that these two comparisons with the laziness technique don't reveal the information of u_2. It is another reason why we set $t = 178 * 2^{-8}$. The comparison between r_3 and $178 * 2^{-8}$ only needs the first 8 bits. If $r_3 \geq 178 * 2^{-8}$, we can determine $r_3 > \ln 2 > u_2$ through the first 8 bits. Otherwise, r_3 is a uniform random number in $[0, 178 * 2^{-8})$. In this case, for the first 8 bits of r_3 and u_2, the probability that they are equal is $1/178$. For the i-th bits of r_3 and u_2 ($i \geq 9$), the probability that they are equal is $1/2$. Therefore the value of u_2 doesn't impact the running time of the two comparisons.

3.3 Applications and Performances

Our exponential Bernoulli sampling algorithm can be applied to Gaussian samplers based on rejection samplings. In this section, we choose FALCON, a lattice-based signature scheme of the third-round candidates in the NIST PQ project, as an example of the application of Algorithm 3.

The Gaussian sampler in FALCON is based on rejection sampling, so it needs to sample an exponential Bernoulli variable $\mathcal{B}_p$ with $p = \exp(-x)$ ($x \geq 0$). Under the parameter sets of FALCON, we can assume $x \in [0, 64 \cdot \ln 2]$. To achieve the security goal of FALCON, the relative error between the ideal and real probabilities can't be greater than 2^{-43} [23,39]. The relative error of our algorithm is not greater than 2^{-48}, which satisfies the requirement of FALCON. In the latest implementations of FALCON, it uses a polynomial approximation $P_{\text{facct}}(x)$ [44] of the exponential function $\exp(x)$ on $[-\ln 2, 0]$ to sample $\mathcal{B}_p$.

In the round 3 NIST package, FALCON contains the optimized implementations at two security levels. There are three implementations at each security level. Because we don't consider the specific instruction set optimization, we choose the ones relying on the IEEE-754 floating-point type to perform our benchmark test. They can work on x68, ARM, and POWER/PowerPC systems.

We apply Algorithm 3 to FALCON and compile the implementations with the compiler option -O3 enabled. The benchmark results are showed in Table 2 and Table 3. The new discrete Gaussian sampling implementation is about 1.19x faster than the original. While the LDL tree has been built upon the secret key, the new signature generation implementation is about 1.15x-1.16x faster than the original. In this paper, all our experiments are performed on a single Intel Core i7-4790 CPU core at 3.6 GHz.

Table 2: Number of Gaussian samples of FALCON per second at 3.6 GHz

	Number of Samples ($\times 10^6$/sec)
Original Implementation	6.31
Our Implementation	7.53

Table 3: Signature generation time of FALCON at 3.6 GHz

Ring Degree	512	1024
Original Implementation	211.27 μs	418.62 μs
Our Implementation	182.58 μs	361.38 μs

The implementation of FALCON doesn't offer the interface to invoke the algorithm for sampling the exponential Bernoulli variable. So we separate this module and compare it with Algorithm 3. We choose the ChaCha20 in OpenSSL and the AES256 counter mode with hardware AES instructions (AES-NI) as the pseudorandom number generators (PRNG) and use g++ 7.5.0 to compile our implementations with the compiler options -O3 and -maes enabled[8]. Table 4 lists the efficiencies of different implementations. If the ChaCha20 in OpenSSL is used as the PRNG, Algorithm 3 is 1.25x faster than the algorithm in FALCON. Algorithm 3 with AES-NI is 1.49x faster than the algorithm in FALCON. The reason is that Algorithm 3 avoids a large number of floating-point multiplications and needs more random numbers. The efficiency of the PRNG has more influence on Algorithm 3 than the algorithm in FALCON.

Table 4: Number of exponential Bernoulli samples per second at 3.6 GHz

PRNG	ChaCha20 ($\times 10^7$/sec)	AES-NI ($\times 10^7$/sec)
Algorithm in FALCON	6.19	6.36
Algorithm 3	7.76	9.47

4 Sampling Integers Using a Discrete Gaussian Distribution with a Small Standard Deviation

In this section, we first introduce our new Gaussian sampler based on rejection sampling, and analyze its correctness and the requirements . Then we use our tool Algorithm 3 to implement our Gaussian sampler and make it isochronous

[8] -maes is only used to compile AES-NI. It can be removed while the implementation doesn't involve AES-NI.

concerning its inputs and output. Finally, we compare our implementations with previous works, and apply it to the lattice cryptography library PALISADE.

4.1 New Discrete Gaussian Sampling Algorithm

We build our Gaussian sampler by employing the rejection sampling strategy of Karney's sampler and present it in Algorithm 4. The rejection sampling strategy of Karney's sampler can be viewed as the generalization of that of the Gaussian sampler in the BLISS signature scheme (BLISS sampler) [11]. The strategy of Karney's sampler is suitable for the distribution with arbitrary $\sigma > \sigma_0$ and c, while the strategy of the BLISS sampler is only suitable for the distribution with σ being an integral multiple of σ_0 and $c = 0$. Du et al. [10] proposed a similar algorithm based on the rejection sampling strategy of Karney's sampler. Although it's faster than Karney's sampler, it doesn't solve the crucial security problem of Karney's Sampler. To make Algorithm 4 resistant to timing attacks and not affecting the security of cryptographic schemes, we need to craft the modules in Algorithm 4, especially the base sampler (step 3) and the algorithm sampling the exponential Bernoulli variable (step 9). In addition, we also use $C(\sigma) \in (0, 1]$ to make σ independent from the rejection rate of Algorithm 4. These are the necessary steps for the practicality of Karney's sampler, which are also our main contributions in this section.

Algorithm 4: New Discrete Gaussian Sampling Algorithm

Input: Standard deviation $\sigma \geq \sigma_0$, center c, parameters $k = \sigma/\sigma_0$, $C(\sigma)$

Output: $z \xleftarrow{\$} D_{\mathbb{Z},\sigma,c}$

1: $f \leftarrow true$
2: **while** $f = true$ **do**
3: $x \xleftarrow{\$} D_{\mathbb{N},\sigma_0}$
4: $y \xleftarrow{\$} \{0, ..., \lceil k \rceil - 1\}$
5: $s \xleftarrow{\$} \{1, -1\}$
6: $z_0 \leftarrow \lceil kx + y + sc \rceil$
7: $d \leftarrow z_0 - (kx + sc)$
8: $p_{rej} \leftarrow C(\sigma) \cdot \exp(-d(d + 2kx)/(2\sigma^2))$
9: $b \xleftarrow{\$} \mathcal{B}_{p_{rej}}$
10: **if** $d < k$ **then**
11: **if** $x \neq 0$ or $d \neq 0$ or $s \neq 1$ **then**
12: **if** $b = 1$ **then**
13: $f \leftarrow false$
14: **end if**
15: **end if**
16: **end if**
17: **end while**
18: $z \leftarrow sz_0$
19: **return** z

To prove the correctness of Algorithm 4, there are two issues to be addressed: (1). After step 10 and step 11 in Algorithm 4, each integer z^* can be calculated by exactly one tuple (x^*, y^*, s^*); (2). We need to show the probability of sampling z^* is exactly proportional to $\rho_{\mathbb{Z},\sigma,c}(z^*)$. Because our rejection sampling strategy is the same as that of Karney's sampler, the idea of proof is implicit in [27,10]. We provide the detailed proof in Appendix B for completeness.

Theorem 6. *The output z sampled by Algorithm 4 is distributed as $D_{\mathbb{Z},\sigma,c}$.*

The floating-point error in Algorithm 4 causes the error of the Bernoulli variable $\mathcal{B}_{p_{rej}}$, so the error of Algorithm 4 stems from the approximate distribution $D^{(R)}_{\mathbb{N},\sigma_0}$ and the approximate Bernoulli variable $\mathcal{B}^{(R)}_{p_{rej}}$, which is the same as FALCON's sampler. We utilize the technique in [23] to estimate the security loss of the cryptographic scheme invoking Algorithm 4 in practice.

Theorem 7. *Let $\lambda^{(I)}$ (resp. $\lambda^{(R)}$) be the security parameter of an implementation using the ideal distributions $D^{(I)}_{\mathbb{N},\sigma_0}$ and $\mathcal{B}^{(I)}_{p_{rej}}$ (resp. the real distributions $D^{(R)}_{\mathbb{N},\sigma_0}$ and $\mathcal{B}^{(R)}_{p_{rej}}$). Let the numbers of sampling from $D_{\mathbb{N},\sigma_0}$ and $\mathcal{B}_{p_{rej}}$ be Q_{bs} and Q_{exp}. If the following conditions are respected, at most two bits of security are lost. In other words, $\lambda^{(I)} - \lambda^{(R)} \leq 2$.*

$$\forall\ p_{rej} \geq 0, \quad \left| \frac{\mathcal{B}^{(R)}_{p_{rej}} - \mathcal{B}^{(I)}_{p_{rej}}}{\mathcal{B}^{(I)}_{p_{rej}}} \right| \leq \sqrt{\frac{\lambda^{(R)}}{Q_{exp}(2\lambda^{(R)}+1)^2}},$$

$$R_{2\lambda^{(R)}+1}(D^{(R)}_{\mathbb{N},\sigma_0}, D^{(I)}_{\mathbb{N},\sigma_0}) \leq 1 + \frac{1}{4Q_{bs}}.$$

In the proof of Theorem 7, we can first bound the Rényi divergence between $\mathcal{B}^{(R)}_{p_{rej}}$ and $\mathcal{B}^{(I)}_{p_{rej}}$ by Lemma 6 and the first condition of Theorem 7. Then, combining Lemma 4 and the upper bound of $R_{2\lambda^{(R)}+1}\left(D^{(R)}_{\mathbb{N},\sigma_0}, D^{(I)}_{\mathbb{N},\sigma_0}\right)$, we can estimate the security loss. We provide the detailed proof in Appendix C.

As suggested by the NIST PQ project, an attacker can make no more than 2^{64} signature or decryption queries. The lattice dimension of a cryptographic scheme is usually less than 2^{12}. Therefore, it's reasonable to assume that the cryptographic scheme calls Algorithm 4 less than 2^{76} in most applications. According to the acceptance probability of Algorithm 4 in Lemma 8, if $\sigma_0 > 1/\sqrt{2\ln 2}$, the expected number of trials of Algorithm 4 is no more than 3. Then we have $Q_{bs} = Q_{exp} \leq 3 \cdot 2^{76} \leq 2^{78}$. If a cryptographic scheme invoking algorithm 4 has 256-bit security, the concrete numerical values of the conditions in Theorem 7 are

$$\sqrt{\frac{\lambda^{(R)}}{Q_{exp}(2\lambda^{(R)}+1)^2}} \approx 2^{-44}, \quad 1 + \frac{1}{4Q_{bs}} \approx 1 + 2^{-80}.$$

4.2 Isochronous Implementations

In this subsection, we first analyze the conditions to make Algorithm 4 isochronous with respect to the standard deviation σ, the center c, and its output z, then introduce how to implement Algorithm 4 and select the appropriate parameters to satisfy those conditions.

To get an isochronous implementation of Algorithm 4, there are five necessary conditions:

1. The base sampler sampling from $D_{\mathbb{N},\sigma_0}$ doesn't leak any information of x.
2. The sampling time of b is independent of $-d(d+2kx)/(2\sigma^2)$.
3. The acceptance probability of Algorithm 4 doesn't rely on c.
4. The sampling time of y is independent of k.
5. The acceptance probability of Algorithm 4 doesn't rely on σ.

The implementation isochronous concerning z and c needs to satisfy the first three conditions. The implementation isochronous concerning z, c, and σ needs to satisfy all five conditions.

For the first condition, we adopt two samplers to instantiate Algorithm 4 respectively. The first one is the CDT sampler. We precompute the approximate cumulative distribution table of $D_{\mathbb{N},1}$ with 80-bit precision shown in Table 5. To produce a sample, we generate a random value r in $[0,1)$ with the same precision and return the index of the last entry in the table that is greater than r. To make the running time isochronous with respect to the output, the CDT sampler has to read the entire table and compare r with each entry. For any $a \leq 512$, our experiment shows that the Rényi Divergence $R_a\left(D_{\mathbb{N},1}^{(R)}, D_{\mathbb{N},1}^{(I)}\right)$ between the real distribution $D_{\mathbb{N},1}^{(R)}$ and the ideal distribution $D_{\mathbb{N},1}^{(I)}$ is bounded by $1+2^{-80}$, which satisfies the second condition in Theorem 7. The advantage of the CDT sampler is that it has a very attractive performance.

Table 5: The approximate CDT of $D_{\mathbb{N},1}$ with 80-bit precision

z	CDT(z) ($\times 2^{-80}$)	z	CDT(z) ($\times 2^{-80}$)
0	519416855270223991024635	5	10517004221616016
1	101208528248637278136991	6	15796660852944
2	7893637264903720998210	7	8733832501
3	23388456691468587 1813	8	1776829
4	2580077773372372849	9	132

Another one is the binary sampler, which is introduced in the BLISS sampler [11]. It sample from the distribution $D_{\mathbb{N},1/\sqrt{2\ln 2}}$. In Algorithm 5, we show how to make the binary sampler isochronous concerning its output. We cut the tail

of the distribution and only sample the non-negative integers no more than 8. Moreover, the while loop in Algorithm 5 never terminates in advance even if the sample has been determined. As each probability of $D_{\mathbb{N},1/\sqrt{2\ln 2}}$ can be accurately represented in binary form, the output distribution $D^{(R)}_{\mathbb{N},1/\sqrt{2\ln 2}}$ does't have a relative error. By some simple calculations and Lemma 5, for any $a \in (1, +\infty)$, we have $R_a\left(D^{(R)}_{\mathbb{N},1/\sqrt{2\ln 2}}, D^{(I)}_{\mathbb{N},1/\sqrt{2\ln 2}}\right) \leq 1+2^{-80}$ satisfying the second condition in Theorem 7. The advantage of the binary sampler is that it doesn't need any precomputation.

Algorithm 5: The Binary Sampler

Output: $z \xleftarrow{\$} D_{\mathbb{Z}^+,1/\sqrt{2\ln 2}}$

```
success ← 0
while success = 0 do
    z ← 0, bad ← 0
    b ←$ {0, 1}
    success ← (b = 0)
    z ← z + (1 − success)
    for i ← 1 to 8 do
        draw random bits b_1...b_{2i−1}
        if b_1...b_{2i−2} ≠ 0...0 then
            bad ← 1
        end if
        if b_{2i−1} = 0 and bad = 0 then
            success ← 1
        end if
        z ← z + (1 − success)
    end for
end while
return z
```

For the second condition, as $C(\sigma) \in (0, 1]$, it's straightforward to sample the exponential Bernoulli variable $\mathcal{B}_{p_{rej}}$ through Algorithm 3. The relative error of Algorithm 3 is no more than 2^{-48} meeting the first condition in Theorem 7.

For the fourth condition, we use Algorithm 6 to sample y. In our implementation, the public parameter h is equal to 32. The acceptance probability of Algorithm 6 is 0.5, so its running time is independent of k.

Both third and fifth conditions are related to the acceptance probability of Algorithm 4. We utilize the technique in FALCON's sampler to satisfy these two conditions. In the following lemma, we prove that $\sigma \geq \eta^+_\epsilon(\mathbb{Z})$ is sufficient for the independence between c and the acceptance probability of Algorithm 4. Moreover, if $\sigma \geq t\sigma_0$, where t, σ_0 are public parameters, and $C(\sigma)$ in Algorithm 4 has an appropriate value, the acceptance probability could not rely on σ.

Algorithm 6: Isochronous Uniform Sampling Algorithm

Input: Parameters $\lceil k \rceil \in (2^{l-1}, 2^l]$ $(l \leq h)$, $\mathrm{P}_k = 2^{l-1}/\lceil k \rceil$

Output: $y \xleftarrow{\$} \{0, ..., \lceil k \rceil - 1\}$

1: $f \leftarrow true$
2: **while** $f = true$ **do**
3: $\quad y \xleftarrow{\$} \{0, ..., 2^h - 1\}$
4: $\quad y \leftarrow y \ \& \ (2^l - 1)$
5: $\quad r \xleftarrow{\$} [0, 1)$
6: $\quad$ **if** $y < \lceil k \rceil$ and $r < \mathrm{P}_k$ **then**
7: $\quad\quad f \leftarrow false$
8: $\quad$ **end if**
9: **end while**
10: **return** y

Lemma 8. *Let $\epsilon \in (0, 1)$, $c \in \mathbb{R}$, $k = \frac{\sigma}{\sigma_0}$ and t be a positive integer. If the standard deviation σ meets the condition that $\sigma \geq \eta_\epsilon^+(\mathbb{Z})$, we can set $C(\sigma) = 1$ and $p_\sigma = \frac{\rho_\sigma(\mathbb{Z})}{2\lceil k \rceil \rho_{\sigma_0}(\mathbb{N})}$. The acceptance probability $P_{\mathrm{true}}(\sigma, c)$ of the while loop in Algorithm 4 satisfies*

$$P_{\mathrm{true}}(\sigma, c) \in p_\sigma \cdot [1 - 2\epsilon, 1].$$

If the standard deviations σ_0, σ meet the condition that $\sigma \geq max\{\eta_\epsilon^+(\mathbb{Z}), t\sigma_0\}$, we can set $C(\sigma) = \frac{t\lceil k \rceil}{(t+1)k} \leq 1$ and $p = \frac{t\sigma_0\sqrt{2\pi}}{2(t+1)\rho_{\sigma_0}(\mathbb{N})}$. The acceptance probability $P_{\mathrm{true}}(\sigma, c)$ of the while loop in Algorithm 4 satisfies

$$P_{\mathrm{true}}(\sigma, c) \in p \cdot [1 - 2\epsilon, 1 + \epsilon].$$

Proof. We use T to represent the set of all tuples (x^*, y^*, s^*) that satisfies the conditions in step 10 and step 11 in Algorithm 4. According to the proof of Theorem 6, each integer z^* can be calculated by exactly one tuple (x^*, y^*, s^*) in set T. Thus, the acceptance probability $P_{true}(\sigma, c)$ of the while loop in Algorithm 4 is:

$$\begin{aligned} P_{true}(\sigma, c) &= \sum_{(x^*, y^*, s^*) \in T} \underbrace{\frac{\rho_{\sigma_0}(x^*)}{\rho_{\sigma_0}(\mathbb{N})}}_{x^* \leftarrow D_{\mathbb{N}, \sigma_0}} \cdot \underbrace{\frac{1}{\lceil k \rceil}}_{y^* \leftarrow \{0, ..., \lceil k \rceil - 1\}} \cdot \underbrace{\frac{1}{2}}_{s^* \leftarrow \{1, -1\}} \cdot \underbrace{C(\sigma) \cdot \frac{\rho_{\sigma, c}(z^*)}{\rho_{\sigma_0}(x^*)}}_{\Pr\{\mathcal{B}_{p_{rej}} = 1\}} \\ &= \sum_{z^* \in \mathbb{Z}} \frac{C(\sigma) \cdot \rho_{\sigma, c}(z^*)}{2\lceil k \rceil \cdot \rho_{\sigma_0}(\mathbb{N})} = \frac{C(\sigma) \cdot \rho_{\sigma, c}(\mathbb{Z})}{2\lceil k \rceil \cdot \rho_{\sigma_0}(\mathbb{N})} \end{aligned}$$

For any $\epsilon \in (0, 1)$, if $\sigma \geq \eta_\epsilon^+(\mathbb{Z}) \geq \eta_\epsilon(\mathbb{Z})$ and $C(\sigma) = 1$, it holds from Lemma 2 that

$$P_{true}(\sigma, c) \in \frac{\rho_\sigma(\mathbb{Z})}{2\lceil k \rceil \cdot \rho_{\sigma_0}(\mathbb{N})} \cdot \left[\frac{1 - \epsilon}{1 + \epsilon}, 1\right] \in \frac{\rho_\sigma(\mathbb{Z})}{2\lceil k \rceil \cdot \rho_{\sigma_0}(\mathbb{N})} \cdot [1 - 2\epsilon, 1],$$

which concludes the first part of Lemma 8.

To make $P_{true}(\sigma, c)$ independent of σ, we need to bound $\rho_\sigma(\mathbb{Z})$. For any $\epsilon \in (0,1)$, if $\sigma \geq \eta_\epsilon^+(\mathbb{Z}) \geq \eta_\epsilon(\mathbb{Z})$, the following relationship holds from Lemma 2 and Lemma 3:

$$P_{true}(\sigma, c) \in \frac{C(\sigma) \cdot \sigma\sqrt{2\pi}}{2\lceil k \rceil \cdot \rho_{\sigma_0}(\mathbb{N})} \cdot [1-2\epsilon, 1+\epsilon].$$

Now, we can use $C(\sigma)$ to eliminate σ and $\lceil k \rceil$. As $k = \frac{\sigma}{\sigma_0}$, t is a positive integer and $\sigma \geq t\sigma_0$, we can set $C(\sigma) = \frac{t\lceil k \rceil}{(t+1)k} \leq 1$. Then, we have:

$$P_{true}(\sigma, c) \in \frac{t\sigma_0\sqrt{2\pi}}{2(t+1)\rho_{\sigma_0}(\mathbb{N})} \cdot [1-2\epsilon, 1+\epsilon],$$

which concludes the second part of Lemma 8. □

In the remaining part of this subsection, we will analyze the influence of the acceptance probability $P_{\text{true}}(\sigma, c)$ on the running time of Algorithm 4 and the adversary advantage. We only consider the scenarios where both σ and c are secret. If σ is public, we can ignore the last two conditions. This means we don't need Algorithm 6 and parameter $C(\sigma)$ to hide σ. In this case, we can utilize a similar method to analyze the success probability of the adversary by the first part of Lemma 8. We omit the details here.

In the following Theorem 8, we show that Algorithm 4 is perfect isochronous with respect to z and statistically isochronous for the Rényi divergence with respect to σ, c.

Theorem 8. *Let $\epsilon \in (0,1)$, $c \in \mathbb{R}$, $k = \frac{\sigma}{\sigma_0}$ and t be an positive integer. Let σ_0, σ be the standard deviations such that $\sigma_0 \in [\frac{1}{\sqrt{2\ln(2)}}, 1]$, $\sigma \geq max\{\eta_\epsilon^+(\mathbb{Z}), t\sigma_0\}$. Let $C(\sigma) = \frac{t\lceil k \rceil}{(t+1)k} \leq 1$, $p = \frac{t\sigma_0\sqrt{2\pi}}{2(t+1)\rho_{\sigma_0}(\mathbb{N})}$ be constants in $(0,1)$. Suppose that the implementation of Algorithm 4 satisfies the first, second, and fourth conditions. Its running time follows a distribution $T(\sigma, c)$ such that:*

$$R_a(T(\sigma, c), T) \lesssim 1 + 4a\epsilon^2 \max\left(\frac{1-p}{p^2}, \frac{1}{1-p}\right) \leq 1 + 24a\epsilon^2$$

for some distribution T independent of σ, c, and z.

In the following Theorem 9, we leverage Theorem 8 to prove that the running time of Algorithm 4 does not help an adversary to break the cryptographic scheme. As is analyzed in [23], we consider that the adversary has access to some function $g(D_{\mathbb{Z},\sigma,c})$ as well as the running time of Algorithm 4. This is intended to capture the fact that in most applications, the output of Algorithm 4 is not given directly to the adversary, but processed by some function g before.

Theorem 9. *Consider an adversary $\mathcal{A}$ making Q_s queries to $g(D_{\mathbb{Z},\sigma,c})$ for some randomized function g, and solving a search problem with success probability $2^{-\lambda}$ for some $\lambda \geq 1$. With the notations of Theorem 8, suppose that $\epsilon \leq \frac{1}{\sqrt{24\lambda Q_s}}$. Learning the running time of each call to Algorithm 4 does not increase the success probability of $\mathcal{A}$ by more than a constant factor.*

To prove Theorem 8, we need to analyze the running time T_0 of one iteration of the while loop in Algorithm 4 and the number $I(\sigma, c)$ of iterations. Let I be the number of iterations in the ideal case. $I(\sigma, c)$ (resp. I) is determined by the probability $P_{true}(\sigma, c)$ (resp. p). As the floating-point operations are isochronous and the first, second, and fourth conditions are satisfied, T_0 follows a distribution independent of σ, c, and z. By Lemma 4, we have $R_a(T(\sigma, c), T) = R_a(I(\sigma, c), I)$. We can deduce the upper bound of $R_a(I(\sigma, c), I)$ from the relation $P_{true}(\sigma, c) \in p \cdot [1 - 2\epsilon, 1 + 2\epsilon]$ and prove the first part of the inequality. The second part can be derived from $p \in [0.34, 0.36]$. The proof of Theorem 9 requires the flexible applications of the three properties in Lemma 4. We put the detailed deductions in Appendix D and Appendix E.

4.3 Applications and Performances

For simplifying the description, we use Type I isochrony and Type II isochrony to clarify whether the implementation hides the information of σ. The two isochronous properties satisfy the requirements of the GPV trapdoor sampler and the MP trapdoor sampler. We offer four different implementations of Algorithm 4. The implementations of Type I isochrony are the same as those of Type II isochrony except utilizing Algorithm 6 and parameter $C(\sigma)$ to hide σ. The base samplers are the CDT sampler and the binary sampler. The binary sampler doesn't need any precomputation, while the CDT sampler need to store the cumulative distribution table in Table 5. Each 80-bit integer is represented by two 64-bit integers in our implementations, so the table consumes $20 \cdot 8 = 160$ bytes. All four implementations need at most several hundred bytes of memory consumption containing other precomputed values such as k and $1/(2\sigma^2)$.

We choose the AES256 counter mode with the AES-NI instruction set as the PRNG and use g++ 7.5.0 to compile the four implementations of Algorithm 4 with the compiler options -O3 and -maes enabled. In each test, we produce 10^4 random centers c in [0,1) for each σ, then generate 10^3 samples with the same σ and c and measure the consumed time. We calculate the benchmark results in Table 6 based on the measured times. The implementations based on the CDT sampler are a little faster than those based on the binary sampler. In the implementations of Type I isochrony, we set the parameter $C(\sigma)$ equal to $(2\lceil k \rceil)/(3k)$ for $\sigma \geq 2\sigma_0$[9]. It causes a significant decline in the performances. If σ is far greater than σ_0, we can adjust $C(\sigma)$ to get better performances. For example, if $\sigma \geq 32\sigma_0$, we can set $C(\sigma)$ equal to $(32\lceil k \rceil)/(33k)$. The implementations based on the binary sampler and the CDT sampler generate 9.68×10^6 and 10.23×10^6 samples per second for $\sigma = 32$.

We summarize the performances of previous works in Table 7. We scale all the numbers to be based on 3.6 GHz. The TwinCDT sampler [30] is isochronous concerning c and z. It offers three different tradeoffs between the running time and the precomputation storage. Although it is at least two times faster than our implementations of Type II isochrony for $\sigma \in [2, 32]$, it's not generic. Its

[9] σ_0 in our implementations is 1 or $1/\sqrt{2 \ln 2}$.

Table 6: Number of samples per second at 3.6 GHz for our new Gaussian sampler Algorithm 4

σ	Type I Isochrony		Type II Isochrony	
	Binary Sampler ($\times 10^6$/sec)	CDT Sampler ($\times 10^6$/sec)	Binary Sampler ($\times 10^6$/sec)	CDT Sampler ($\times 10^6$/sec)
2	6.97	7.13	9.91	13.62
8	6.70	7.12	10.96	13.72
32	6.77	7.20	11.20	13.55
2^{15}	6.76	7.16	11.54	13.99
2^{20}	6.74	7.32	11.34	14.14

Table 7: Summary of previous works at 3.6 GHz

Algorithm	σ	Memory (KB)	Number of Samples ($\times 10^6$/sec)
Twin-CDT Sampler [30]	2	1.4/4.6/46	43.73/53.53/65.51
Twin-CDT Sampler [30]	8	3/10/100	32.31/45.69/54.44
Twin-CDT Sampler [30]	32	9.5/32/318	29.47/34.08/36.51
Convolution Sampler [34]	2^{15}	$2^{5.4}$	10.59 (online) 1.53 (online+offline)
Karney's Sampler [27]	$1 - 2^{20}$	< 1	8.10
DWZ Sampler [10]	$4 - 2^{20}$	< 1	13.97
FALCON's Sampler [23]	$1.29 - 1.82$	< 1	7.53

precomputation storage increases significantly along with σ increasing. The convolution sampler [34] satisfies Type I isochrony. Our implementations of Type I isochrony are at least four times faster than the convolution sampler. If we only measure the time during the online phase, our implementations will generate the base samples during the offline phase. We can assume $\sigma \geq 13 > 4\sqrt{2}\eta_\epsilon(\mathbb{Z})$ for reasonable ϵ just as the benchmark test in [34][10]. With $C(\sigma) = (13\lceil k \rceil)/(14k)$, our implementations of Type I isochrony generate 12.43×10^6 and 13.11×10^6 samples per second for $\sigma = 2^{15}$ during the online phase, which are faster than the convolution sampler.

The DWZ sampler [10] and Karney's sampler could not resist timing attacks, though the DWZ sampler is faster than our implementations. FALCON's sampler [23] satisfies Type I isochrony. It's faster than our implementations of Type I isochrony. However, its storage and running time increase rapidly along with the maximum standard deviation increasing. In addition, if the minimum standard deviation is larger than 2, our implementations will have better performances.

We apply Algorithm 4 to the lattice cryptography library PALISADE [8] and use its latest version as of this writing[11] to evaluate the performances. We replace

[10] The smoothing parameter in [34] is $\sqrt{2\pi}$ times that in this paper.
[11] https://gitlab.com/palisade/palisade-development/-/tree/release-v1.10.6

Table 8: Running time of the G-lattice sampling algorithm in PALISADE at 3.6 GHz

	Running Time (ms)
Original Implementation	3.74
Our Implementation	2.09

Karney's sampler in PALISADE with our implementations based on the CDT sampler to sample from the Gaussian distributions over the integers. PALISADE uses the BLAKE2 hash function[12] as the PRNG. In PALISADE, Karney's sampler generates 2.49×10^6 samples per second, while our implementations generate 5.92×10^6 and 3.21×10^6 samples.

PALISADE contains the implementations [22] of IBE [32] and CP-ABE [43]. Both rely on the latest MP trapdoor sampler [18]. We mainly focus on the G-lattice sampling algorithm in the MP trapdoor sampler. It determines the running time during the online phase. All standard deviations are public in the G-lattice sampling algorithm, so we make it invoke the implementation of Type II isochrony. The parameters include the modulus q, the dimension n of lattice, the base b for the gadget lattice G, and the standard deviation σ. Let $(q, n, b, \sigma) = (12289, 256, 2, 100)$. The benchmark results are shown in Table 8. Our implementation is 1.79x faster than the original.

5 Sampling Integers Using a Normal Distribution

In this section, we first introduce the rejection sampling strategy of the new COSAC sampler, then analyze its correctness and security requirements. Finally, we implement it to confirm the theoretical analysis and compare it with Algorithm 4.

5.1 New COSAC Sampler

The COSAC sampler [44] uses the normal distribution $\mathcal{N}(0, \sigma)$ and the rejection sampling to sample from the distributions $D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ with $c_F \in [-1/2, 1/2]$. The original rejection sampling strategy is independent of c_F and requires about two trials on average to output a sample. Our new rejection sampling strategy varies with c_F. The expected number of trials of the new sampler is about half of that of the original.

The original COSAC sampler utilizes two normal distributions $\mathcal{N}_1(0, \sigma)$ and $\mathcal{N}_2(0, \sigma)$ to sample from the distributions $D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ with $c_F \in [-1/2, 1/2]$. $\mathcal{N}_1(0, \sigma)$ and $\mathcal{N}_2(0, \sigma)$ are used to sample the positive and negative integers respectively. In each trial, the sampler chooses $\mathcal{N}_1(0, \sigma)$ or $\mathcal{N}_2(0, \sigma)$ with equal probability. Assume $\mathcal{N}_1(0, \sigma)$ is chosen and x^* is sampled from $\mathcal{N}_1(0, \sigma)$. If there exists a positive integer z^* such that $x^* \in (z^* - 3/2, z^* - 1/2]$, the positive integer z^* is accepted with the probability $\rho_{\sigma,c_F}(z^*)/\rho_\sigma(x^*)$. Assume $\mathcal{N}_2(0, \sigma)$ is chosen

[12] https://www.blake2.net

and x^* is sampled from $\mathcal{N}_2(0,\sigma)$. If there exists a negative integer z^* such that $x^* \in [z^*+1/2, z^*+3/2)$, the negative integer z^* is accepted with the probability $\rho_{\sigma,c_F}(z^*)/\rho_\sigma(x^*)$. No matter which normal distribution is chosen, the trial fails with a probability of about 0.5.

The new COSAC sampler only needs one normal distribution $\mathcal{N}(0,\sigma)$ to sample from the distributions $D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ with $c_F \in [-1/2, 1/2]$. In each trial, the sampler sample x^* from $\mathcal{N}(0,\sigma)$. If $x^*+c_F >= 0$ and $x^* \in [z^*-1-c_F, z^*-c_F)$, the positive integer z^* is accepted with the probability $\rho_{\sigma,c_F}(z^*)/\rho_\sigma(x^*)$; If $x^*+c_F < 0$ and $x^* \in [z^*-c_F, z^*+1-c_F)$, the negative integer z^* is accepted with the probability $\rho_{\sigma,c_F}(z^*)/\rho_\sigma(x^*)$.

Our sampler is represented in Algorithm 8. It invokes Algorithm 7 to sample from the distributions $D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ with $c_F \in [-1/2, 1/2]$. The correctness of Algorithm 8 and Algorithm 7 is followed by our analysis above and some simple integral calculations. Theorem 10 indicates that the acceptance probability of Algorithm 7 converges to 1 as σ increases. In theory, Algorithm 8 is almost two times faster than the original COSAC sampler.

Algorithm 7: Rounding Sampler

Input: Standard deviation σ, center $c_F \in [-1/2, 1/2]$

Output: $z \xleftarrow{\$} D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$

$f \leftarrow true$
while $f = true$ **do**
  $x_0 \xleftarrow{\$} \mathcal{N}(0,1)$
  $x \leftarrow \sigma x_0$
  $z_0 \leftarrow x + c_F$
  $b_0 \leftarrow (z_0 \geq 0)$
  $z \leftarrow \lfloor z_0 \rfloor + b_0$
  $p_{rej} \leftarrow \exp((x^2 - (z - c_F)^2)/(2\sigma^2))$
  $b \xleftarrow{\$} \mathcal{B}_{p_{rej}}$
  if $b = 1$ **then**
    $f \leftarrow false$
  end if
end while
return z

Theorem 10. *The outputs of Algorithm 7 and Algorithm 8 are distributed as $D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ and $D_{\mathbb{Z},\sigma,c}$ respectively. Let $\epsilon \in (0,1)$, and $\sigma > \eta_\epsilon(\mathbb{Z})$, the acceptance probability $P_{true}(\sigma, c_F)$ of Algorithm 7 satisfies*

$$P_{true}(\sigma, c_F) \geq 1 - \epsilon - \frac{1}{\sigma\sqrt{2\pi}}.$$

Due to the impossibility of achieving perfect distributions in practice, we need Theorem 11 to estimate the security loss of the cryptographic schemes invoking

Algorithm 8: New COSAC Sampler

Input: Standard deviation σ, center $c \in \mathbb{R}$, normalization factor $S = \rho_{\sigma,c}(\mathbb{Z})$

Output: $z \xleftarrow{\$} D_{\mathbb{Z},\sigma,c}$

$c_I \leftarrow \lfloor c \rceil$
$c_F \leftarrow c - c_I$
$p_0 \leftarrow \exp(-c_F^2/(2\sigma^2))/S$
$b \xleftarrow{\$} \mathcal{B}_{p_0}$
if $b = 1$ **then**
    $z \leftarrow 0$
else
    $z \xleftarrow{\$} D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$
end if
return $z + c_I$

Algorithm 8. The error of Algorithm 8 stems from the approximate distribution $\mathcal{N}^{(R)}(0,1)$ and the approximate Bernoulli variables $\mathcal{B}_{p_0}^{(R)}$, $\mathcal{B}_{p_{rej}}^{(R)}$. The proof of Theorem 11 is similar to that of Theorem 7, except estimating the influence of the continuous distribution $\mathcal{N}(0,1)$. We provide the detailed proofs of Theorem 10 and Theorem 11 in Appendix F and Appendix G.

Theorem 11. *Let $\lambda^{(I)}$ (resp. $\lambda^{(R)}$) be the security parameter of an implementation using the ideal distributions $\mathcal{N}^{(I)}(0,1)$ and $\mathcal{B}_p^{(I)}$ (resp. the real distributions $\mathcal{N}^{(R)}(0,1)$ and $\mathcal{B}_p^{(R)}$). Let the numbers of sampling from $\mathcal{N}(0,1)$ and $\mathcal{B}_p$ be Q_{bs} and Q_{exp}, the absolute error of the sample values between $\mathcal{N}^{(I)}(0,1)$ and $\mathcal{N}^{(R)}(0,1)$ be $\Delta_{\mathcal{N}}$. If the following conditions are respected, at most two bits of security are lost. In other words, $\lambda^{(I)} - \lambda^{(R)} \leq 2$.*

$$\forall\ p \geq 0, \quad \left| \frac{\mathcal{B}_p^{(I)} - \mathcal{B}_p^{(R)}}{\mathcal{B}_p^{(I)}} \right| \leq \sqrt{\frac{\lambda^{(R)}}{Q_{exp}(2\lambda^{(R)}+1)^2}},$$

$$\Delta_{\mathcal{N}} \leq \frac{1}{2\sigma\sqrt{(4\lambda^{(R)}+2)Q_{bs}}}.$$

5.2 Performance of New COSAC Sampler

We implement Algorithm 8 by modifying the implementation of the original COSAC sampler. The implementation utilizes AVX2 intrinsic instructions to improve the performance and employ the Box-Muller continuous Gaussian sampler [25] to sample from $\mathcal{N}(0,1)$. The AES256 counter mode with the ASE-NI instructions is used as the PRNG. For convenience, we use g++ 7.5.0 to compile the implementations with the same compiler options -O3 -march=native enable as [44]. The benchmark results are shown in Table 9.

In Table 9, the average numbers of trials are basically consistent with the theoretical results. However, the new COSAC sampler is 1.46x-1.63x faster than

Table 9: Comparison between the original COSAC sampler and the new COSAC sampler at 3.6 GHz

σ	Number of Samples ($\times 10^6$/sec)		Average Number of Trials	
	Original	New	Original	New
2	6.81	11.07	2.49	1.24
8	8.23	12.20	2.10	1.05
32	8.73	13.09	2.03	1.01
2^{15}	8.89	13.19	2.00	1.00
2^{20}	8.94	13.23	2.00	1.00

the original. The main reason is the additional operations to hide z. As claimed in [44], our implementation of Algorithm 8 may also reveal σ. Although the implementation involves AVX2 intrinsic instructions, we compare it with the implementations of Type II isochrony in Table 6. The new COSAC sampler is faster than the implementation based on the binary sampler but slower than the implementation based on the CDT sampler. It is noteworthy that the absolute error of the Box-Muller continuous Gaussian sampler is no more than 2^{-48}. For $\sigma \in [2, 2^{20}]$ and $\lambda^{(R)} = 256$, the provable security is accessible only for $Q_{bs} \leq 2^{44}$ based on the second condition in Theorem 11.

6 Conclusions

In this work, we mainly proposed three algorithms to design a generic, secure and efficient Gaussian sampling algorithm over the integers. They are Algorithm 3, Algorithm 4, and Algorithm 8. Algorithm 3 is to sample the exponential Bernoulli variables. Compared with the polynomial approximation method, our algorithm reduces the floating-point multiplications significantly. We applied it to FALCON to decrease the signature generation time. One interesting phenomenon is that the polynomial approximation method can benefit from the Haswell microarchitecture developed by Intel as the fourth-generation core. If the compiler options include -march=haswell, the running time of the polynomial approximation method will decrease by about 45% on a single Intel Core i7-4790 CPU at 3.6 GHz. It's an important issue how to improve the performance of our algorithm on the specific architecture.

Algorithm 4 can sample from the Gaussian distributions over the integers while hiding the standard deviation, center, and output. It can be used as the fundamental operation modular in the lattice cryptography library. We applied it to PALISADE and obtained a more secure and efficient implementation of the MP trapdoor sampler. We can utilize the constant-time implementation of the Knuth-Yao sampler [26] as the base sampler and improve the performances of our implementations further. Besides, Theorem 9 is only available for search

problems. For decision problems, one may prove the security by the techniques in [35].

Algorithm 8 also samples from the Gaussian distributions over the integers. It's not as competitive as Algorithm 4 so far. However, it has the lowest expected number of trials among the algorithms based on rejection sampling. It's necessary to find a more efficient algorithm sampling from the normal distribution to improve the performance. In addition, it also deserves further research to make the acceptance probability of Algorithm 8 independent of the standard deviation and center to resist timing attacks.

References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT. pp. 553–572. Springer (2010)
2. Ahrens, J.H., Dieter, U.: Extensions of Forsythe's method for random sampling from the normal distribution. Mathematics of Computation **27**(124), 927–937 (1973)
3. Ajtai, M.: Generating hard instances of lattice problems. In: Miller, G.L. (ed.) STOC. pp. 99–108. ACM (1996)
4. Bai, S., Langlois, A., Lepoint, T., Stehlé, D., Steinfeld, R.: Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT. pp. 3–24. Springer (2015)
5. Barthe, G., Belaïd, S., Espitau, T., Fouque, P., Rossi, M., Tibouchi, M.: Galactics: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) CCS. pp. 2147–2164. ACM (2019)
6. Bootle, J., Delaplace, C., Espitau, T., Fouque, P.A., Tibouchi, M.: LWE without modular reduction and improved side-channel attacks against BLISS. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT. pp. 494–524. Springer (2018)
7. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) STOC. pp. 575–584. ACM (2013)
8. Cousins, D.B., Crescenzo, G.D., Gür, K.D., King, K., Polyakov, Y., Rohloff, K., Ryan, G.W., Savas, E.: Implementing conjunction obfuscation under entropic ring LWE. In: IEEE Symposium on Security and Privacy. pp. 354–371. IEEE Computer Society (2018)
9. Dai, W., Doröz, Y., Polyakov, Y., Rohloff, K., Sajjadpour, H., Savas, E., Sunar, B.: Implementation and evaluation of a lattice-based key-policy ABE scheme. IEEE Trans. Inf. Forensics Secur. **13**(5), 1169–1184 (2018)
10. Du, Y., Wei, B., Zhang, H.: A rejection sampling algorithm for off-centered discrete Gaussian distributions over the integers. Sci. China Inf. Sci. **62**(3), 39103:1–39103:3 (2019)
11. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO. pp. 40–56. Springer (2013)
12. Ducas, L., Lyubashevsky, V., Prest, T.: Efficient identity-based encryption over NTRU lattices. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT. pp. 22–41. Springer (2014)

13. Ducas, L., Nguyen, P.Q.: Faster Gaussian lattice sampling using lazy floating-point arithmetic. In: Wang, X., Sako, K. (eds.) ASIACRYPT. pp. 415–432. Springer (2012)
14. van Erven, T., Harremoës, P.: Rényi divergence and kullback-leibler divergence. IEEE Trans. Inf. Theory **60**(7), 3797–3820 (2014)
15. Espitau, T., Fouque, P.A., Gérard, B., Tibouchi, M.: Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) CCS. pp. 1857–1874. ACM (2017)
16. Forsythe, G.E.: Von Neumann's comparison method for random sampling from the normal and other distributions. Mathematics of Computation **26**(120), 817–826 (1972)
17. Fouque, P., Kirchner, P., Tibouchi, M., Wallet, A., Yu, Y.: Key recovery from gram-schmidt norm leakage in hash-and-sign signatures over NTRU lattices. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT. pp. 34–63. Springer (2020)
18. Genise, N., Micciancio, D.: Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT. pp. 174–203. Springer (2018)
19. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) STOC. pp. 169–178. ACM (2009)
20. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork, C. (ed.) STOC. pp. 197–206. ACM (2008)
21. Groot Bruinderink, L., Hülsing, A., Lange, T., Yarom, Y.: Flush, gauss, and reload – A cache attack on the BLISS lattice-based signature scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES. pp. 323–345. Springer (2016)
22. Gür, K.D., Polyakov, Y., Rohloff, K., Ryan, G.W., Sajjadpour, H., Savas, E.: Practical applications of improved Gaussian sampling for trapdoor lattices. IEEE Trans. Computers **68**(4), 570–584 (2019)
23. Howe, J., Prest, T., Ricosset, T., Rossi, M.: Isochronous Gaussian sampling: From inception to implementation. In: Ding, J., Tillich, J.P. (eds.) PQC. pp. 53–71. Springer (2020)
24. Hu, Y., Jia, H.: A new Gaussian sampling for trapdoor lattices with arbitrary modulus. Des. Codes Cryptogr. **87**(11), 2553–2570 (2019)
25. Hülsing, A., Lange, T., Smeets, K.: Rounded Gaussians - fast and secure constant-time sampling for lattice-based crypto. In: Abdalla, M., Dahab, R. (eds.) PKC. pp. 728–757. Springer (2018)
26. Karmakar, A., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Pushing the speed limit of constant-time discrete gaussian sampling. A case study on the falcon signature scheme. In: DAC. p. 88. ACM (2019)
27. Karney, C.F.F.: Sampling exactly from the normal distribution. ACM Trans. Math. Softw. **42**(1), 3:1–3:14 (2016)
28. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehle, D., Bai, S.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2020), available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`
29. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT. pp. 1–23. Springer (2010)
30. Melchor, C.A., Ricosset, T.: CDT-based Gaussian sampling: From multi to double precision. IEEE Trans. Computers **67**(11), 1610–1621 (2018)

31. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In: FOCS. pp. 356–365. IEEE Computer Society (2002)
32. Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT. pp. 700–718. Springer (2012)
33. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. In: FOCS. pp. 372–381. IEEE Computer Society (2004)
34. Micciancio, D., Walter, M.: Gaussian sampling over the integers: Efficient, generic, constant-time. In: Katz, J., Shacham, H. (eds.) CRYPTO. pp. 455–485. Springer (2017)
35. Micciancio, D., Walter, M.: On the bit security of cryptographic primitives. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT. pp. 3–28. Springer (2018)
36. Peikert, C.: An efficient and parallel Gaussian sampler for lattices. In: Rabin, T. (ed.) CRYPTO. pp. 80–97. Springer (2010)
37. Pessl, P., Bruinderink, L.G., Yarom, Y.: To BLISS-B or not to be: Attacking strongswan's implementation of post-quantum signatures. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) CCS. pp. 1843–1855. ACM (2017)
38. Prest, T.: Sharper bounds in lattice-based cryptography using the Rényi divergence. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT. pp. 347–374. Springer (2017)
39. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2020), available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`
40. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) STOC. pp. 84–93. ACM (2005)
41. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehle, D.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2020), available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`
42. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)
43. Zhang, J., Zhang, Z.: A ciphertext policy attribute-based encryption scheme without pairings. In: Wu, C., Yung, M., Lin, D. (eds.) Inscrypt. pp. 324–340. Springer (2011)
44. Zhao, R.K., Steinfeld, R., Sakzad, A.: COSAC: compact and scalable arbitrary-centered discrete Gaussian sampling over integers. In: Ding, J., Tillich, J.P. (eds.) PQC. pp. 284–303. Springer (2020)
45. Zhao, R.K., Steinfeld, R., Sakzad, A.: FACCT: fast, compact, and constant-time discrete Gaussian sampler over integers. IEEE Trans. Computers **69**(1), 126–137 (2020)

A Proof of Lemma 3

Proof. By the Fourier transform of $\rho_\sigma(x)$ and the Poisson summation formula, we have: $\rho_\sigma(\mathbb{Z}) = \sigma\sqrt{2\pi} \cdot \left(1 + 2\sum_{i\geq 1} \exp(-2i^2\pi^2\sigma^2)\right)$. If $\epsilon \in (0,1)$ and $\sigma > \eta_\epsilon^+(\mathbb{Z})$, the inequality holds from Lemma 1: $\exp\left(-2\pi^2\sigma^2\right) \leq \epsilon/(2(1+\epsilon))$. Combining

these two relations with $\epsilon \in (0,1)$, we can complete the proof

$$\begin{aligned}\rho_\sigma(\mathbb{Z}) &= \sigma\sqrt{2\pi}\cdot\left(1+2\sum_{i\geq 1}\exp(-2i^2\pi^2\sigma^2)\right)\\ &\leq \sigma\sqrt{2\pi}\cdot\left(1+\sum_{i\geq 1}2\cdot\left(\frac{\epsilon}{2(1+\epsilon)}\right)^{i^2}\right)\leq \sigma\sqrt{2\pi}\cdot(1+\epsilon).\end{aligned}$$

□

B Proof of Theorem 6

Proof. We first prove that each integer z^* is calculated by exactly one tuple (x^*, y^*, s^*). We can write c as $c = c_1 + c_2$ in which $c_1 \in [0,1)$ and $c_2 \in \mathbb{Z}$. Then we have

$$d = \lceil kx + sc_1\rceil + y - (kx + sc_1), \quad z = s\cdot(\lceil kx + sc_1\rceil + y) + c_2.$$

If $c_1 \neq 0$, $z \leq c_2$ for $s = -1$, and $z > c_2$ for $s = 1$. However, if $c_1 = 0$, $z = c_2$ can be calculated by two tuples (0, 0, -1) and (0, 0, 1). The step 11 makes z greater than c_2 for $s = 1$ no matter whether c_1 is equal to 0, which solves the problem. Next, we analyze that each integer $z^* > c_2$ can be calculated by exactly one tuple (x^*, y^*, s^*) for $s = 1$. For any $x \geq 0$, $\lceil k\rceil - 1 \leq \lceil k(x+1) + sc_1\rceil - \lceil kx + sc_1\rceil \leq \lceil k\rceil$. As $y \in \{0, ..., \lceil k\rceil - 1\}$, each integer $z^* > c_2$ can be calculated by at least one tuple (x^*, y^*, s^*). However, if some x_{bad} satisfies that $\lceil k(x_{bad}+1) + sc_1\rceil - \lceil kx_{bad} + sc_1\rceil = \lceil k\rceil - 1$, then $z = \lceil k(x_{bad}+1) + sc_1\rceil + c_2$ could be calculated by two tuples $(x_{bad}, \lceil k\rceil - 1, 1)$ and $(x_{bad}+1, 0, 1)$. Due to the fact that $\lceil k(x+1) + sc_1\rceil > \lceil kx + sc_1\rceil + \lceil k\rceil - 1$ is equivalent to $k(x+1) + sc_1 > \lceil kx + sc_1\rceil + \lceil k\rceil - 1$, the tuple $(x_{bad}, \lceil k\rceil - 1, 1)$ is rejected after the step 10. Then, each integer $z^* > c_2$ is calculated by exactly one tuple. For $s = -1$, it's similar to prove that each integer $z^* <= c_2$ is also calculated by exactly one tuple.

Now, we calculate the probability that each integer z^* is sampled. Assume that z^* is calculated by the tuple (x^*, y^*, s^*), i.e. $z^* = s^*\cdot\lceil kx^* + y^* + s^*c\rceil$. Let $z_0^* = \lceil kx^* + y^* + s^*c\rceil$ and $d^* = z_0^* - (kx^* + s^*c)$, then we have:

$$\begin{aligned}\Pr[z = z^*] &\propto \exp(-\frac{x^{*2}}{2\sigma_0^2})\cdot C(\sigma)\exp(-\frac{d^*(d^*+2kx^*)}{2\sigma^2})\\ &= C(\sigma)\exp(-\frac{(z_0^* - s^*c)^2}{2\sigma^2})\\ &= C(\sigma)\exp(-\frac{(z^* - c)^2}{2\sigma^2}) \propto D_{\mathbb{Z},\sigma,c}(z^*),\end{aligned}$$

which concludes the proof. □

C Proof of Theorem 7

Proof. We first define three different cases:

1. (Ideal Case) The implementation uses two ideal distributions $D^{(I)}_{\mathbb{N},\sigma_0}$ and $\mathcal{B}^{(I)}_{p_{rej}}$.
2. (Intermediate Case) The implementation uses a real distribution $D^{(R)}_{\mathbb{N},\sigma_0}$ and an ideal distribution $\mathcal{B}^{(I)}_{p_{rej}}$.
3. (Real Case) The implementation uses two real distributions $D^{(R)}_{\mathbb{N},\sigma_0}$ and $\mathcal{B}^{(R)}_{p_{rej}}$.

We recall that $\lambda^{(I)}$ (resp. $\lambda^{(R)}$) is the security parameter of the Ideal (resp. Real) Case. We aim at computing $\Delta\lambda = \lambda^{(I)} - \lambda^{(R)}$.

We set the order $a = 2\lambda^{(R)} + 1$. Let ϵ_I (resp. ϵ_{IN}, ϵ_R) be the probability that the adversary breaks the scheme in the use of the Ideal (resp. Intermediate, Real) Case. By data processing inequality and probability preservation of the Rényi divergence in Lemma 4:

$$\epsilon_I \geq \epsilon_{IN}^{\frac{a}{a-1}} / R_a \left(D^{(R)}_{\mathbb{N},\sigma_0}, D^{(I)}_{\mathbb{N},\sigma_0} \right)^{Q_{bs}},$$

$$\epsilon_{IN} \geq \epsilon_{R}^{\frac{a}{a-1}} / R_a \left(\mathcal{B}^{(R)}_{p_{rej}}, \mathcal{B}^{(I)}_{p_{rej}} \right)^{Q_{exp}}.$$

By definition, $\epsilon_R = 2^{-\lambda^{(R)}}$, so we can deduce the relationship between ϵ_I and ϵ_R using $\epsilon_R^{\frac{a}{a-1}} = \epsilon_R / \sqrt{2}$:

$$\epsilon_{IN} \geq \epsilon_R / \left(\sqrt{2} \cdot R_a \left(\mathcal{B}^{(R)}_{p_{rej}}, \mathcal{B}^{(I)}_{p_{rej}} \right)^{Q_{exp}} \right),$$

$$\epsilon_I \geq \epsilon_R / \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot R_a \left(\mathcal{B}^{(R)}_{p_{rej}}, \mathcal{B}^{(I)}_{p_{rej}} \right)^{\frac{aQ_{exp}}{a-1}} \cdot R_a \left(D^{(R)}_{\mathbb{N},\sigma_0}, D^{(I)}_{\mathbb{N},\sigma_0} \right)^{Q_{bs}} \right).$$

So we have:

$$\Delta\lambda \leq \log \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot R_a \left(\mathcal{B}^{(R)}_{p_{rej}}, \mathcal{B}^{(I)}_{p_{rej}} \right)^{\frac{aQ_{exp}}{a-1}} \cdot R_a \left(D^{(R)}_{\mathbb{N},\sigma_0}, D^{(I)}_{\mathbb{N},\sigma_0} \right)^{Q_{bs}} \right). \quad (5)$$

Based on the first condition in Theorem 7 and $a = 2\lambda^{(R)} + 1$, an application of Lemma 6 yields to

$$R_a \left(\mathcal{B}^{(R)}_{p_{rej}}, \mathcal{B}^{(I)}_{p_{rej}} \right) \leq 1 + \frac{a-1}{4aQ_{exp}}. \quad (6)$$

By combining (5) (6) and the second condition in Theorem 7, we get

$$\begin{aligned} \Delta\lambda &\leq \log \left(\sqrt{2}^{\frac{a}{a-1}+1} \left(1 + \frac{a-1}{4aQ_{exp}} \right)^{\frac{aQ_{exp}}{a-1}} \left(1 + \frac{1}{4Q_{bs}} \right)^{Q_{bs}} \right) \\ &\leq \log \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot \exp(1/4)^2 \right) \leq \log \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot 2 \right) \leq 2, \end{aligned}$$

where the second inequality is deduced from $\left(1 + \frac{x}{n}\right)^n \leq \exp(x)$ for $x, n > 0$. □

D Proof of Theorem 8

The number of iterations of the while loop in Algorithm 4 follows a geometric distribution of its acceptance probability. When proving Theorem 8, we will need the following lemma [23] to bound the Rényi divergence between two geometric distributions.

Lemma 9 *Let $\mathcal{P}$ and $\mathcal{Q}$ be geometric distributions of parameters $p, q \in (0,1)$. Suppose there exists $\delta = o(1/(a+1))$ such that:*

$$\exp(-\delta) \le p/q \le \exp(\delta),$$

$$\exp(-\delta) \le (1-p)/(1-q) \le \exp(\delta).$$

Then the Rényi divergence between $\mathcal{P}$ and $\mathcal{Q}$ is bounded as follows:

$$R_a(\mathcal{P}, \mathcal{Q}) \lesssim 1 + \frac{a(1-p)\delta^2}{p^2} \left(\sim 1 + \frac{a(1-q)\delta^2}{q^2} \right).$$

Now we can prove Theorem 8.

Proof. Let T_0 denote the running time of one iteration of the while loop in Algorithm 4. If the floating-point operations are isochronous, the basic sampler is isochronous with respect to its output and the algorithms sampling b and y is isochronous with respect to their inputs and outputs, then T_0 follows a distribution which is independent of σ, c, z.

Let $I(\sigma, c)$ (resp. I) denote the number of iterations of the while loop when each iteration accepts with probability $P_{true}(\sigma, c)$ (resp. p). $I(\sigma, c)$ (resp. I) is a geometric distribution of parameter $P_{true}(\sigma, c)$ (resp. p). By Lemma 9, we have $P_{true}(\sigma, c) \in p \cdot [1-\epsilon, 1+2\epsilon]$. Through a few simple computations, we can get the following inequalities:

$$1 - 2\epsilon \le \frac{P_{true}(\sigma, c)}{p} \le 1 + 2\epsilon,$$

$$1 - \frac{p}{1-p} \cdot 2\epsilon \le \frac{1 - P_{true}(\sigma, c)}{1-p} \le 1 + \frac{p}{1-p} \cdot 2\epsilon.$$

Let $\delta = 2\epsilon \cdot \max(1, \frac{p}{1-p})$. If ϵ is small enough, it follows from Lemma 9 that:

$$R_a(I(\sigma, c), I) \lesssim 1 + \frac{a(1-p)\delta^2}{p^2} \lesssim 1 + 4a\epsilon^2 \cdot \max\left(\frac{1-p}{p^2}, \frac{1}{1-p} \right)$$

The total running time T (resp. $T(\sigma, c)$) of Algorithm 4 is a function of T_0 and I (resp. $I(\sigma, c)$) for some function f. This allows to apply the data-processing inequality:

$$\begin{aligned} R_a(T(\sigma, c), T) &= R_a(f(T_0, I(\sigma, c)), f(T_0, I)) \\ &\le R_a(I(\sigma, c), I) \\ &\lesssim 1 + 4a\epsilon^2 \cdot \max\left(\frac{1-p}{p^2}, \frac{1}{1-p} \right). \end{aligned}$$

Since $\sigma_0 \in [\frac{1}{\sqrt{2\ln 2}}, 1]$ and $p = \frac{t\sigma_0\sqrt{2\pi}}{2(t+1)\rho_{\sigma_0}(\mathbb{N})} \in [0.34, 0.36]$, we can get the final conclusion:

$$R_a(T(\sigma, c), T) \lesssim 1 + 4a\epsilon^2 \max\left(\frac{1-p}{p^2}, \frac{1}{1-p}\right) \leq 1 + 24a\epsilon^2.$$

□

E Proof of Theorem 9

Proof. Let D denote the output distribution of $g(D_{\mathbb{Z},\sigma,c})$. In the real (resp. ideal) case, we can consider without loss of generality that the adversary can query the joint distribution $(D, T(\sigma, c))$ (resp. (D, T)), where $T(\sigma, c)$ (resp. T) is the running time of Algorithm 4 in the real (resp. ideal) case. In the proof of Theorem 8, we have shown that both T and $T(\sigma, c)$ are independent of the output z. Thus, both T and $T(\sigma, c)$ are independent of the distribution D. Let $a = \lambda$, and P_0, P_1 denote the success probabilities of $\mathcal{A}$ in the ideal and real cases, respectively. Since $P_0 = 2^{-\lambda}$ and $\epsilon \leq \frac{1}{\sqrt{24\lambda Q_s}}$, it holds from Lemma 4 and Theorem 8 that:

$$\begin{aligned}
P_1 &\leq \left(P_0 \cdot R_a((D, T(\sigma, c))^{Q_s}, (D, T)^{Q_s})\right)^{\frac{a-1}{a}} \\
&\leq \left(P_0 \cdot R_a((D, T(\sigma, c)), (D, T))^{Q_s}\right)^{\frac{a-1}{a}} \\
&\leq \left(P_0 \cdot R_a(T(\sigma, c), T)^{Q_s}\right)^{\frac{a-1}{a}} \\
&\lesssim \left(P_0 \cdot \left(1 + 24a\epsilon^2\right)^{Q_s}\right)^{\frac{a-1}{a}} \\
&\lesssim \left(P_0 \cdot \left(1 + \frac{1}{Q_s}\right)^{Q_s}\right)^{\frac{\lambda-1}{\lambda}} \\
&\lesssim 2^{-(\lambda-1)} \cdot e,
\end{aligned}$$

which concludes the proof. □

F Proof of Theorem 10

Proof. To prove that the output of Algorithm 7 is distributed as $D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$, we need to calculate the probability that each non-zero integer z^* is sampled. If $z^* > 0$, z^* is calculated from x^* in the interval $[z^* - 1 - c_F, z^* - c_F)$, then we

can obtain the probability of z^*

$$\begin{aligned}\Pr[z = z^* | z^* > 0] = &\int_{z^*-1-c_F}^{z^*-c_F} \underbrace{\frac{\rho_\sigma(x^*)}{\sigma\sqrt{2\pi}}}_{x^* \leftarrow \mathcal{N}(0,\sigma)} \cdot \underbrace{\frac{\rho_{\sigma,c_F}(z^*)}{\rho_\sigma(x^*)}}_{\Pr\{\mathcal{B}_{p_{rej}}=1\}} dx \\ =&\int_{z^*-1-c_F}^{z^*-c_F} \frac{\rho_{\sigma,c_F}(z^*)}{\sigma\sqrt{2\pi}} dx \\ =&\frac{\rho_{\sigma,c_F}(z^*)}{\sigma\sqrt{2\pi}} \propto D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}(z^*).\end{aligned}$$

If $z^* < 0$, z^* is calculated from x^* in the interval $[z^* - c_F, z^* + 1 - c_F)$, then it's easy to check that $\Pr[z = z^* | z^* \in \mathbb{Z}, z^* < 0] \propto D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}(z^*)$. So, for any non-zero integer z^*, we can conclude that $\Pr[z = z^*] \propto D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}(z^*)$.

In Algorithm 8, $\Pr[z = 0] = \exp(-c_F^2/(2\sigma^2))/S = D_{\mathbb{Z},\sigma,c_F}(0)$. Therefore, variable z is distributed as $D_{\mathbb{Z},\sigma,c_F}$. Since $c = c_I + c_F$, $z + c_I$ is distributed as $D_{\mathbb{Z},\sigma,c}$.

Now, let's estimate the acceptance probability $P_{true}(\sigma, c_F)$ of Algorithm 7:

$$\begin{aligned}P_{true}(\sigma, c_F) &= \Pr[z = z^* | z^* > 0] + \Pr[z = z^* | z^* < 0] \\ &= \sum_{i=1}^{\infty} \left(\frac{\rho_{\sigma,c_F}(i)}{\sigma\sqrt{2\pi}} + \frac{\rho_{\sigma,c_F}(-i)}{\sigma\sqrt{2\pi}} \right) \\ &= \frac{\rho_{\sigma,c_F}(\mathbb{Z}\setminus\{0\})}{\sigma\sqrt{2\pi}} \\ &= \frac{\rho_{\sigma,c_F}(\mathbb{Z}) - \rho_{\sigma,c_F}(0)}{\sigma\sqrt{2\pi}} \\ &\geq 1 - \epsilon - \frac{1}{\sigma\sqrt{2\pi}},\end{aligned}$$

where the last inequality is deduced from Lemma 2 and $\rho_{\sigma,c_F}(0) \leq 1$. □

G Proof of Theorem 11

Proof. The proof of Theorem 11 is very similar to that of Theorem 7. The only difference is that we need to evaluate the impact of the error of the normal distribution $\mathcal{N}(0, 1)$ on security. We first define three different cases:

1. (Ideal Case) The implementation uses two ideal distributions $\mathcal{N}^{(I)}(0, 1)$ and $\mathcal{B}_p^{(I)}$.
2. (Intermediate Case) The implementation uses a real distribution $\mathcal{N}^{(R)}(0, 1)$ and an ideal distribution $\mathcal{B}_p^{(I)}$.
3. (Real Case) The implementation uses two real distributions $\mathcal{N}^{(R)}(0, 1)$ and $\mathcal{B}_p^{(I)}$.

Our goal is to compute $\Delta\lambda = \lambda^{(I)} - \lambda^{(R)}$. Let the order a of the Rényi divergence be $2\lambda^{(R)} + 1$. Let ϵ_I (resp. ϵ_{IN}, ϵ_R) be the probability that the adversary breaks the scheme in the use of the Ideal (resp. Intermediate, Real) Case. Let $D^{(I)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ (resp. $D^{(IN)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ [13]) be the output distribution of Algorithm 7 in the use of the the Ideal (resp. Intermediate) Case. By data processing inequality and probability preservation of the Rényi divergence in Lemma 4:

$$\epsilon_I \geq \epsilon_{IN}^{\frac{a}{a-1}} / R_a \left(D^{(IN)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}, D^{(I)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F} \right)^{Q_{bs}},$$

$$\epsilon_{IN} \geq \epsilon_R^{\frac{a}{a-1}} / R_a \left(\mathcal{B}_p^{(R)}, \mathcal{B}_p^{(I)} \right)^{Q_{exp}}.$$

By the definitions, we have $\epsilon_R = 2^{-\lambda^{(R)}}$, $\epsilon_I = 2^{-\lambda^{(I)}}$ and $\epsilon_R^{\frac{a}{a-1}} = \epsilon_R/\sqrt{2}$, then

$$\Delta\lambda \leq \log \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot R_a \left(\mathcal{B}_p^{(R)}, \mathcal{B}_p^{(I)} \right)^{\frac{aQ_{exp}}{a-1}} \cdot R_a \left(D^{(IN)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}, D^{(I)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F} \right)^{Q_{bs}} \right).$$

Based on the second condition of Theorem 11, we can bound the relative error between $D^{(IN)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ and $D^{(I)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$. For any positive integer z^*,

$$\begin{aligned}
&\left| \frac{D^{(IN)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}(z^*) - D^{(I)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}(z^*)}{D^{(I)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}(z^*)} \right| \\
\leq &\left| \frac{\int_{z^*-1-c_F-\sigma\Delta_{\mathcal{N}}}^{z^*-c_F+\sigma\Delta_{\mathcal{N}}} \frac{\rho_{\sigma,c_F}(z^*)}{\sigma\sqrt{2\pi}}\, dx - \frac{\rho_{\sigma,c_F}(x^*)}{\sigma\sqrt{2\pi}}}{\frac{\rho_{\sigma,c_F}(x^*)}{\sigma\sqrt{2\pi}}} \right| \\
= &2\sigma\Delta_{\mathcal{N}} \leq \frac{1}{\sqrt{(4\lambda^{(R)}+2)Q_{bs}}}.
\end{aligned}$$

The same result holds for any negative integer. Therefore, the relative error between $D^{(IN)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ and $D^{(I)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ is no more than $\frac{1}{\sqrt{(4\lambda^{(R)}+2)Q_{bs}}}$.

By combining Lemma 6 and the bounds of the relative errors, we can get:

$$R_a \left(D^{(IN)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}, D^{(I)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F} \right) \leq 1 + \frac{1}{4Q_{bs}},$$

$$R_a \left(\mathcal{B}_p^{(R)}, \mathcal{B}_p^{(I)} \right) \leq 1 + \frac{a-1}{4aQ_{exp}}.$$

Thus, we have

$$\begin{aligned}
\Delta\lambda &\leq \log \left(\sqrt{2}^{\frac{a}{a-1}+1} \left(1 + \frac{a-1}{4aQ_{exp}} \right)^{\frac{aQ_{exp}}{a-1}} \left(1 + \frac{1}{4Q_{bs}} \right)^{Q_{bs}} \right) \\
&\leq \log \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot \exp(1/4)^2 \right) \leq 2,
\end{aligned}$$

concluding the proof. □

[13] $D^{(IN)}_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ is not the real output distribution of Algorithm 7, because the Bernoulli distribution is perfect in the Intermediate Case.