Generic, Efficient and Isochronous Gaussian Sampling over the Integers

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

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, 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. In particular, it can't be avoided in trapdoor sampling until now. However, it's still a challenging work how to construct a generic, efficient, and isochronous Gaussian sampler. 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 13.66%-15.52%.
Second, we develop a new 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'18), an open-source lattice cryptography library. The new implementation of trapdoor sampling in PALISADE has better performance while resisting timing attacks.
Third, we improve the efficiency of the COSAC sampler (PQC'20). 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 [16], and has conjectured security against quantum computers [40,34]. Most lattice-based cryptographic schemes are based on two main average-case problems, the short integer solution (SIS) problem [2], the learning with errors (LWE) problem [38], and their analogs over rings [28,26]. Discrete Gaussian distributions are at the core of security reduction proofs from

the worst-case lattice problems to the average-case problems [30,6]. 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 [18,14,35,5] against the Gaussian sampler in the BLISS signature [10]. For this reason, some schemes replace Gaussian distributions with other distributions [25,39], 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 [17,33,29]. Trapdoor sampling can be used to construct many powerful cryptographic applications, including signature [17,37], (hierarchical) identity-based encryption ((H)IBE) [17,1,12], and attribute-based encryption (ABE) [41], etc. There are two extensively used trapdoor sampling algorithms at present. The first one is proposed by Gentry, Peikert, and Vaikuntanathan [17] (GPV trapdoor sampler), while the second one is proposed by Micciancio and Peikert [29] (MP trapdoor sampler). Both of them require to sample from the Gaussian distributions over the integers with different standard deviations and arbitrary centers [12,37,15,21,11]. To resist timing attacks, the former should be isochronous concerning the standard deviations, centers, and outputs, while the latter should be isochronous concerning the centers and outputs. In some lattice cryptography libraries, such as PALISADE [7,8,19], it's one of the fundamental building blocks to sample from the Gaussian distributions over the integers. So it's important to propose a Gaussian sampler over the integers which are generic, efficient, and isochronous.

To sample from the distributions with different standard deviations and arbitrary centers, there are two strategies to design 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 [31] belongs to the former. It's easy to make its running time independent of the standard deviations and centers. However, its efficiency highly relies on online/offline skills, and its whole running time is not competitive. For the Gaussian samplers based on rejection samplings, the most important problem is how to retain the security and generality at the same time. Karney's sampler [24], the COSAC sampler [42], and FALCON's sampler [20] are three recently proposed Gaussian samplers based on rejection samplings. Karney's sampler doesn't consider how to resist timing attacks, while the running time of the COSAC sampler may reveal the standard deviations. FALCON's sampler can resist timing attacks, but it's only suitable for the distributions with small standard deviations. One noticeable feature of Gaussian samplers based on rejection samplings is that they always need to sample an exponential Bernoulli variable $\mathcal{B}_p$ with a probability $p = \exp(-x)$ $(x \geq 0)$ being true. It's an important time-consuming module, and a mainstream solution is approximating the exponential function by a polynomial [43,4].

To implement the trapdoor sampling algorithm, the lattice cryptography library PALISADE [7] 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 generic, efficient, and isochronous Gaussian sampler over the integers.

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 basic distribution and design a new Gaussian sampler based on rejection sampling. The COSAC sampler is also a Gaussian sampler based on rejection sampling. It uses the normal distribution as the basic distribution. We reduce the expected number of trials of the COSAC sampler to improve its 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. Compared with the mainstream method approximating the exponential function by a polynomial [43], our algorithm has the following advantages:

- Our algorithm has higher efficiency while retaining the security in rejection sampling scenes. On the one hand, we derive the improvement of performance from avoiding computing the exponential function and reducing floating-point operations significantly. On the other hand, because the sampling values in rejection sampling scenes are public and revealed by the execution of the algorithm, and the running time of our sampling algorithm is independent of its input, our algorithm could resist timing attacks.
- Our algorithm is essentially more straightforward to be implemented with only integer operations and has better numerical stability.
- We apply our algorithm to FALCON [37], 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 has been built upon the secret key, the signature generation time of the new implementation decreases $13.66\% - 15.52\%$ than the original.

We derive the idea of our new Gaussian sampler from Karney's sampler [24] and FALCON's sampler [20]. We employ the Rényi divergence [3,36], a tool which in recent years is used in the security proofs of many lattice-based cryptographic schemes, to prove that our sampling algorithm resists timing attacks. Specifically,

we show that our sampling algorithm is isochronous [20] with respect to its inputs (i.e. standard deviation and center) and output (i.e. the sampling value). Our discrete Gaussian sampling algorithm has the following advantages:

- Our Gaussian sampler can sample from the distributions with different standard deviations and arbitrary centers.
- Our Gaussian sampler can adaptively hide the information of standard deviation or center in different applications. Additionally, when our sampler hides the information of both of them, we could adjust the parameter $C(\sigma)$ in Algorithm 4 according to different minimum standard deviations required by different cryptographic schemes, such that our sampler has the best performance.
- 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 decrease by 44.12%.

The core of our new COSAC sampler is to adjust the rejection sampling strategy according 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, the expected number of trials of the new sampler converges to 1, which is the best in theory for the algorithms based on rejection samplings. However, the new COSAC sampler is only 1.46x-1.63x faster than the original due to the additional operations to hide the center and output. It is not faster than our new Gaussian sampler and needs a more efficient algorithm sampling from the normal distribution to improve its performance.

2 Preliminaries

2.1 Notations

Let $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{N}$ be the set of real numbers, integers and non-negative integers respectively. We use the notation $\mathcal{B}_p$ to denote the Bernoulli distribution with parameter p. For a distribution D, we use the notation $x \leftarrow D$ to mean that x is chosen according to the distribution D. If S is a set, then $x \leftarrow S$ means that x is sampled uniformly at random from S. 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 Smooth Parameter

The smooth parameter is a lattice quantity proposed by Micciancio and Regev [30]. For $\epsilon > 0$, the smooth 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. In the literature, some definitions of the smooth parameter scale our definition by a factor $\sqrt{2\pi}$.

Lemma 1 ([30]). *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 ([30,17]). *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 may need the following lemma to make the running time of a discrete Gaussian sampling algorithm independent of σ. This lemma is implicit in [20].

Lemma 3 ([20], implicit in Lemma 7). *For any $\epsilon > 0$, $\sigma > \eta_\epsilon^+(\mathbb{Z})$, we have:*

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

To handle the infinite domain of the distribution $D_{\mathbb{Z},\sigma,c}$, the analysis and design of the algorithms usually take advantage of the rapid decay of Gaussian distributions to approximate the original distribution $D_{\mathbb{Z},\sigma,c}$ by sampling from a finite domain. The next lemma is useful in determining the tailcut parameter.

Lemma 4 ([17]). *For any $\epsilon > 0$, $\sigma \geq \eta_\epsilon(\mathbb{Z})$, $c \in \mathbb{R}$ and $t > 0$, we have:*

$$\Pr_{x \leftarrow D_{\mathbb{Z},\sigma,c}}[|x - c| \geq t\sigma] \leq 2\exp\left(-\frac{t^2}{2}\right) \cdot \frac{1+\epsilon}{1-\epsilon}.$$

2.4 Rényi Divergence

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

Definition 1 ([3]). *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

$$R_\infty(\mathcal{P}, \mathcal{Q}) = \max_{x \in Supp(\mathcal{P})} \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

The Rényi divergence is not a distance, as it is neither symmetric nor does it verify the triangle inequality, which makes it less convenient than the statistical distance. On the other hand, it does verify cryptographically useful properties. We recall some important properties of the Rényi divergence from [3].

Lemma 5 ([3]). *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 $(\mathcal{P}_i)_i, (\mathcal{Q}_i)_i$,*
$$R_a(\prod_i \mathcal{P}_i, \prod_i \mathcal{Q}_i) = \prod_i R_a(\mathcal{P}_i, \mathcal{Q}_i).$$
- ***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 distribution and the ideal distribution.

Lemma 6 ([36]). *Let $\mathcal{P}$ and $\mathcal{Q}$ be two distributions such that Supp($\mathcal{P}$) $\subseteq$ Supp($\mathcal{Q}$). For any $x \in Supp(\mathcal{P})$, 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 the algorithm involves floating-point numbers, we usually use the double type or 64-bit integers to approximate floating-point numbers, which results in the relative error between the real distribution and the ideal distribution. The following lemma shows that a bound of δ on the relative error between two distributions implies a bound $O(\delta^2)$ on the log of the Rényi divergence.

Lemma 7 ([36]). *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 ([20]). *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 $\mathcal{A}$ 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 $\mathcal{D}$.

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

Proof. We first analyze the probability that n is odd in step 8 of Algorithm 1 for any $x_2 \in [0, 1)$. If $n = n_0$ in step 8, there are $n_0 + 1$ variables $u_1, u_2, ..., u_{n_0+1}$ uniformly sampled from [0, 1) in step 3 and step 7. 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, 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

Algorithm 1: Von Neumann's Algorithm

Output: A variable x from the exponential distribution

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

probability that $n = n_0$ in step 8 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 8 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 precision. Then we show how to design an isochronous algorithm in the particular application. Finally, we apply it to the FALCON signature.

3.1 Basic Exponential Bernoulli Sampling Algorithm

Our exponential Bernoulli sampling algorithm is derived from von Neumann's algorithm presented in Algorithm 1. One important step of proving the correctness of Algorithm 1 is to show that the probability that n is even after the while loop equals $\exp(-x_2)$. Inspired by the proof, we can sample a Bernoulli variable with probability $\exp(-x)$ being true utilizing the step 3 - step 7 in Algorithm 1 for any $x \in [0,1)$. In order to sample 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: $b \leftarrow \mathcal{B}_{\exp(-x)}$
1: $u_1 \leftarrow \lfloor x/\ln 2 \rfloor$, $u_2 \leftarrow x - u_1 \cdot \ln 2$
2: $r_1 \leftarrow \{0, 1, ..., 2^{u_1} - 1\}$
3: $b_1 \leftarrow (r_1 = 0)$
4: $n \leftarrow 0$
5: $r_2 \leftarrow u_2$, $r_3 \leftarrow [0, 1)$
6: **while** $r_2 > r_3$ **do**
7: $\quad n \leftarrow n + 1$
8: $\quad r_2 \leftarrow r_3$, $r_3 \leftarrow [0, 1)$
9: **end while**
10: $b_2 \leftarrow (n \ is \ even)$
11: $b \leftarrow b_1 \ \& \ b_2$
12: **return** b

When we apply Algorithm 2 to a cryptographic scheme, it is necessary to bound the relative error of Algorithm 2 to quantify the security loss of the cryptographic scheme after implementing Algorithm 2 by the floating-point number. Theorem 1 explains the relationship between the relative error of Algorithm 2 and the precision of its floating-point numbers.

Theorem 1. *Let $P^{(R)}$ be the probability that the variable b of the real algorithm is true where the floating numbers of Algorithm 2 keep λ bits after the decimal point. Let $P^{(I)}$ be the probability that the variable b of the ideal algorithm is true. Then the relative error between $P^{(R)}$ and $P^{(I)}$ satisfies that*

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

Proof. Obviously, if λ is infinite, Algorithm 2 is an ideal Bernoulli sampling algorithm. So we only need to analyze the difference between the finite case (i.e. the real algorithm) and the infinite case (i.e. the ideal algorithm). 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 $P_2^{(I)}$ can be divided into two parts Δ_1 and Δ_2. Δ_1 stems from the approximation of u_2 while Δ_2 is incurred by the approximate sampling of r_3. As u_2 retains λ decimal places, the difference between the approximate number $u_2^{(R)}$ and the accurate number $u_2^{(I)}$ is not great than $2^{-\lambda}$. So we have

$$\begin{aligned} \frac{|\Delta_1|}{P_2^{(I)}} &= \left| \frac{\exp(-u_2^{(R)}) - \exp(-u_2^{(I)})}{\exp(-u_2^{(I)})} \right| = \left| \exp(-(u_2^{(R)} - u_2^{(I)})) - 1 \right| \\ &\approx \left| u_2^{(R)} - u_2^{(I)} \right| \leq 2^{-\lambda}. \end{aligned} \tag{2}$$

To estimate the upper bound of Δ_2, we will analyze the process of the while loop (step 6 - step 9) of Algorithm 2. Because both r_2 and r_3 retain λ decimal places and r_3 is sampled uniformly from [0,1), the probability that the condition of the while loop can be determined by comparing the approximate numbers $r_2^{(R)}$ and $r_3^{(R)}$ is exactly $1 - 2^{-\lambda}$. For the i-th while loop ($i \geq 1$), we denote A_i as the event $r_2^{(R)} = r_3^{(R)}$, B_i as the event $r_2^{(R)} > r_3^{(R)}$ and C_i as the event $r_2^{(R)} < r_3^{(R)}$. Event B_{i-1} must have happened in the i-th loop ($i \geq 2$).

If event A_i doesn't happen for any $i \geq 1$, the precision of $r_3^{(R)}$ will not influence the process of the while loop. Our goal is to calculate the probability sum of all A_i which is the upper bound of Δ_2. Because $r_2^{(R)}$ is always no more than $u_2^{(R)} \leq u_2^{(I)} + 2^{\lambda}$ and $\Pr\{A_i\} = \Pr\{B_{i-1}\} \cdot 2^{-\lambda}$, we have

$$\Pr\{A_i\} \leq \begin{cases} 2^{-\lambda} & (i = 1) \\ \left(u_2^{(I)} + 2^{-\lambda}\right)^{i-1} \cdot 2^{-\lambda} & (i \geq 2) \end{cases}$$

As $u_2^{(I)} \in [0, \ln(2))$ and $2^{-\lambda}$ is much smaller than $\ln(2)$, we can give a upper bound of $(\Delta_2 / P_2^{(I)})$:

$$\frac{\Delta_2}{P_2^{(I)}} \leq \frac{\sum_{i=1}^{\infty} \Pr\{A_i\}}{\exp\left(-u_2^{(I)}\right)} \leq 2^{-\lambda+1} \cdot \sum_{i=0}^{\infty} \left(u_2^{(I)} + 2^{-\lambda}\right)^i \leq 2^{-\lambda+3}. \tag{3}$$

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

$$\frac{\left|P^{(R)} - P^{(I)}\right|}{P^{(I)}} = \frac{\left|P_2^{(R)} - P_2^{(I)}\right|}{P_2^{(I)}} = \frac{|\Delta_1 + \Delta_2|}{P_2^{(I)}} \leq 2^{-\lambda} + 2^{-\lambda+3},$$

which concludes the proof. □

In our implementation of Algorithm 2, the double type is used in the floating-point operations and 64-bit integers are used to approximate the floating numbers r_2, r_3 , so λ is at least 52. Then the relative error of Algorithm 2 is not more than

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

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 $u_2 < \ln 2$. □

Laziness Technique. If we use 64-bit integers to represent the random numbers r_1 and r_3, there are less than three 64-bit integers sampled on average during each run of Algorithm 2. Those sampled integers r_1, r_3 are only involved in the comparison operations, so we can employ the laziness technique to speed up Algorithm 2. The laziness technique has been introduced in some previous works [13,20]. Its key observation is that when at least one of the two compared 64 bits 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 integer in 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 is 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 27 in one running process of Algorithm 2 with this optimization.

3.2 Isochronous Exponential Bernoulli Sampling Algorithm

Timing attack is a common attack for cryptographic schemes. As the floating-point divisions rarely offer constant-time execution guarantees [42], we can precompute $\ln 2$ and $(1/\ln 2)$ such that Algorithm 2 only involves the floating-point subtractions and multiplications. However, it still seems to be difficult to make Algorithm 2 resistant against timing attacks without sacrificing efficiency significantly. Algorithm 2 can be divided into two parts. The first and second parts sample the Bernoulli variables b_1 and b_2 with probabilities 2^{-u_1} and $\exp(-u_2)$ to be true respectively. The first part is easy to be implemented against timing

attacks without sacrificing efficiency significantly, while it's difficult for the second part. Because the value of b_2 is closely related to the sampling number of r_3 which determines the algorithm running time.

To overcome the obstacle and retain the efficiency, we restrict the application of Algorithm 2 to the rejection sampling algorithm. In the rejection sampling algorithm, different values of b will lead to the different executions of the branch statement. The execution process will reveal the value of b directly. The output b of Algorithm 2 is not a sensitive variable at all. There is no need to make the value of b_2 independent of the algorithm running time. We only focus on how to make the input x independent of the running time, then we will obtain an isochronous exponential Bernoulli sampling algorithm in the rejection sampling scenario. Fortunately, it's easy to reach this goal.

Theorem 3. *Suppose that the floating-point additions, subtractions and multiplications are isochronous. Algorithm 3 is perfectly isochronous with respect to its input x. What's more, for $u_2 \in [0, \ln 2)$, in one running process of Algorithm 3, the expected sampling number S of r_3 is 2, which is independent of u_2.*

Proof. In Algorithm 3, the input x is divided into u_1 and u_2. If we prove that u_1 and u_2 are independent of the running time of Algorithm 3, Algorithm 3 is perfectly isochronous. u_1 and u_2 are used to sample the variables b_1 and b_2 respectively. The sample of b_1 consists of one uniform sampling of r_1 and one comparison between r_1 and zero, which is independent of u_1 clearly. 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 left problem is whether the sampling number of r_3 is independent of u_2. The expected sampling number S of r_3 of Algorithm 3 is

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

which is independent of u_2. □

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 our algorithm.

FALCON contains a discrete Gaussian sampling algorithm based on rejection sampling, so it needs to sample from a Bernoulli variable with a probability $p = \exp(-x)$ $(x \geq 0)$ being true. To achieve the security goal of FALCON, the relative error between the ideal probability and the real probability can't be greater than 2^{-43} [20,37]. The relative error of our algorithm is not greater than 2^{-48}, which satisfies the requirement of FALCON.

Algorithm 3: Isochronous Exponential Bernoulli Sampling Algorithm

Input: $x \geq 0$
Output: $b \leftarrow \mathcal{B}_{\exp(-x)}$

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

In order to sample the exponential Bernoulli variable, FALCON reduces the parameter x modulo $\ln 2$ such that $x = u_1 \ln 2 + u_2$, then computes $\exp(-x) = 2^{-u_1} \exp(-u_2)$. FALCON uses a polynomial approximation of the exponential function $\exp(x)$ on $[-\ln 2, 0]$ to compute $\exp(-u_2)$. In the latest constant-time implementations of FALCON, it uses the polynomial approximation $P_{\text{facct}}(x)$ provided in FACCT [42] to sample the exponential Bernoulli variable. FALCON offers three different implementations. The first one is a generic implementation utilizing double types, while the other two implementations are designed for specific platforms.

We aim to design a generic and efficient exponential Bernoulli sampling algorithm, so we haven't considered the specific instruction set optimization. We apply Algorithm 3 to the first implementation of FALCON and compile the new implementation with the default compiler flags in the Makefile. The benchmark results are showed in Table 1 and Table 2. The new discrete Gaussian sampling implementation is about 1.24x faster than the original. While the LDL tree has been built upon the secret key, the new signature generation implementation is about 1.15x-1.18x 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.

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[3]. In each test, we generate 10^4 samples (with a uniform random parameter x in $[0, 1)$) for 10^4 times and measure the consumed time. Finally, we calculate the benchmark

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

Table 1: Number of Gaussian samples of FALCON per second at 3.6 GHz

	Number of Samples ($\times 10^6$/sec)
Original Implementation	6.98
Our Implementation	8.69

Table 2: Signature generation time of FALCON at 3.6 GHz

Degree	256	512	1024
Original Implementation	96.65 μs	193.96 μs	384.59 μs
Our Implementation	82.59 μs	163.86 μs	332.05 μs

results in Table 3 based on the measured times. If the ChaCha20 in OpenSSL is used as the PRNG, Algorithm 3 is 1.24x faster than the algorithm in FALCON. If AES-NI is used as the PRNG, Algorithm 3 is 1.48x 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. So the efficiency of the PRNG has a greater impact on Algorithm 3 than the algorithm in FALCON.

Table 3: Number of exponential Bernoulli samples per second at 3.6 GHz

PRNG	ChaCha20 ($\times 10^7$/sec)	AES-NI ($\times 10^7$/sec)
Implementation in FALCON	6.01	6.08
Algorithm 3	7.45	9.01

4 Sampling Integers Based on 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 security requirements. Then we use our tool Algorithm 3 to implement our Gaussian sampler and make it isochronous with respect to its inputs and output. Finally, we compare our implementations with the existing algorithms, and apply it to the lattice cryptography library PALISADE.

4.1 New Discrete Gaussian Sampling Algorithm

We present our sampling algorithm in Algorithm 4. The algorithm needs a basic sampler to sample from the discrete Gaussian distribution $D_{\mathbb{N},\sigma_0}$ with the small

standard deviation σ_0. The parameter $C(\sigma)$ could make Algorithm 4 isochronous with respect to σ. Our algorithm can be viewed as the general case of the discrete Gaussian sampling algorithm in the BLISS signature scheme (BLISS sampler) [10]. It can sample from the discrete Gaussian distribution with $\sigma \geq \sigma_0$ and arbitrary center c, while the BLISS sampler can only sample from the distribution with σ being an integral multiple of σ_0 and $c = 0$. In fact, Algorithm 4 is essentially the same as the BLISS sampler if σ is an integral multiple of σ_0 and $c = 0$. The rejection sampling strategy of Algorithm 4 is implied in Karney's sampler [24]. Karney's sampler doesn't consider how to resist timing attacks. However, we can make Algorithm 4 isochronous with respect to its inputs and output.

Algorithm 4: New Discrete Gaussian Sampling Algorithm

Input: Standard deviation $\sigma \geq \sigma_0$, center c, parameters $k = \sigma/\sigma_0$, $C(\sigma)$
Output: $z \leftarrow D_{\mathbb{Z},\sigma,c}$

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

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

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. □

Algorithm 4 is based on rejection sampling and a distribution with a fixed standard deviation, which is similar to FALCON's sampler [20]. In practice, no one could achieve perfect distributions. We can only achieve the approximate distribution $D_{\mathbb{N},\sigma_0}^{(R)}$ and sample the approximate Bernoulli variable $\mathcal{B}_{p_{rej}}^{(R)}$. As analyzed in [20], if the number Q_z a cryptographic scheme invokes Algorithm 4 is limited, its security loss is tolerable under certain conditions on $D_{\mathbb{N},\sigma_0}^{(R)}$ and $\mathcal{B}_{p_{rej}}^{(R)}$.

Theorem 5. *Let $\lambda^{(I)}$ (resp. $\lambda^{(R)}$) be the security parameter of an implementation using the ideal distributions $D_{\mathbb{N},\sigma_0}^{(I)}$ and $\mathcal{B}_{p_{rej}}^{(I)}$ (resp. the real distributions $D_{\mathbb{N},\sigma_0}^{(R)}$ and $\mathcal{B}_{p_{rej}}^{(R)}$). 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, $\lambda^{(I)} - \lambda^{(R)} \leq 2$.*

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

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

In the proof of Theorem 5, we can first bound the Rényi divergence between $\mathcal{B}_{p_{rej}}^{(R)}$ and $\mathcal{B}_{p_{rej}}^{(I)}$ by Lemma 7 and the first condition of Theorem 5. Then, combining Lemma 5 and the upper bound of $R_{2\lambda^{(R)}+1}\left(D_{\mathbb{N},\sigma_0}^{(R)}, D_{\mathbb{N},\sigma_0}^{(I)}\right)$, we can estimate the security loss. We provide the detailed proof in Appendix A.

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 $Q_z \leq 2^{76}$ in most applications of Algorithm 4. According to the acceptance probability of Algorithm 4 in Lemma 9, 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 5 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, the first requirement is that the floating-point operations should be isochronous. We need to precompute $1/\sigma$ to avoid the floating-point divisions which rarely offer constant-time execution guarantees. There are other five necessary conditions:

1. The basic 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 needs to satisfy the first two conditions. 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 basic samplers to instantiate Algorithm 4 respectively. The first one is the cumulative distribution table (CDT) sampler. We precompute the approximate cumulative distribution table of $D_{\mathbb{N},1}$ with 80-bit precision shown in Table 4. 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 $R_a(D_{\mathbb{N},1}^{(R)}, D_{\mathbb{N},1}^{(I)})$ between the real distribution $D_{\mathbb{N},1}^{(R)}$ and the ideal distribution

$D^{(I)}_{\mathbb{N},1}$ is bounded by $1+2^{-80}$, which satisfies the second condition in Theorem 5. The advantage of the CDT sampler is that it has a very attractive performance.

Table 4: 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	789363726490372099821 0	7	8733832501
3	233884566914685871813	8	1776829
4	2580077773372372849	9	132

Another one is the binary sampler, which is introduced in the BLISS sampler [10]. 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 6, for any $a \in (1,+\infty)$, we have $R_a(D^{(R)}_{\mathbb{N},1/\sqrt{2\ln 2}}, D^{(I)}_{\mathbb{N},1/\sqrt{2\ln 2}}) \leq 1+2^{-80}$ satisfying the second condition in Theorem 5. The advantage of the binary sampler is that it doesn't need any precomputation.

For the second condition, we can utilize Algorithm 3 described in subsection 3.2 to sample the exponential Bernoulli variable $\mathcal{B}_{p_{rej}}$. The relative error of Algorithm 3 is no more than 2^{-48} meeting the first condition in Theorem 5.

For the fourth condition, we designed Algorithm 6 to sample y. 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 a sufficient condition of the independence between c and the acceptance probability of Algorithm 4, and if the parameter $C(\sigma)$ in Algorithm 4 has an appropriate value, the acceptance probability of Algorithm 4 could not rely on σ.

Lemma 9. *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*

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

Algorithm 5: The Binary Sampler

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

```
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
```

Algorithm 6: Isochronous Uniform Sampling Algorithm

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

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

```
f ← true
while f = true do
    y ← {0, ..., 2^l − 1}
    r ← [0, 1)
    if y < ⌈k⌉ and r < P_k then
        f ← false
    end if
end while
return y
```

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

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

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 4, 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:

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

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

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

which concludes the first part of Lemma 9.

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:

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

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:

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

which concludes the second part of Lemma 9. □

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 9. We omit the details here.

In the following Theorem 6, 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 6. *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 floating-point additions, subtractions, multiplications, and rounding operations are isochronous. Besides, 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 7, we leverage Theorem 6 to prove that the running time of our implementations of Algorithm 4 does not help an adversary to break the cryptographic scheme. As is analyzed in [20], 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 7. *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 6, 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 6, 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 5, 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 $P_{true}(\sigma, c) \in p \cdot [1 - \epsilon, 1 + 2\epsilon]$ 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 7 requires the flexible applications of the three properties in Lemma 5. We put the detailed deductions in Appendix B and Appendix C.

4.3 Applications and Performances

We offer four different implementations of Algorithm 4. The first two implementations are isochronous concerning the center c and its output z. Their basic samplers are the CDT sampler and the binary sampler respectively. The last two implementations are isochronous concerning the standard deviation σ, center c, and output z. They are the same as the first two implementations except utilizing Algorithm 6 and parameter $C(\sigma)$ to hide σ. The binary sampler doesn't need any precomputation, while the CDT sampler need to store the cumulative distribution table in Table 4. 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 $1/(2\sigma^2)$.

Table 5: Number of samples per second at 3.6 GHz for our new Gaussian sampler Algorithm 4 isochronous concerning c and z

σ	Binary Sampler ($\times 10^6$/sec)	CDT Sampler ($\times 10^6$/sec)
2	9.91	13.62
8	10.96	13.72
32	11.20	13.55
2^{15}	11.54	13.99
2^{20}	11.34	14.14

Table 6: Number of samples per second at 3.6 GHz for our new Gaussian sampler Algorithm 4 isochronous concerning σ, c, and z

σ	Binary Sampler ($\times 10^6$/sec)	CDT Sampler ($\times 10^6$/sec)
2	6.97	7.11
8	6.70	7.11
32	6.77	7.14
2^{15}	6.76	7.18
2^{20}	6.74	7.33

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 5 and Table 6 based on the measured times.

In Table 5 and Table 6, the implementations based on the CDT sampler are a little faster than that based on the binary sampler. The principal reason is that the expected number of trials of the binary sampler is about 1.3 times that of the CDT sampler. In Table 6, we set the parameter $C(\sigma)$ equal to $(2\lceil k \rceil)/(3k)$ which means $\sigma \geq 2\sigma_0$. This causes a significant decline in the performances of the last two implementations. 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 [27] is isochronous concerning c and z, so we compare it with the implementations of Table 5. The TwinCDT sampler offers three different tradeoffs between the running time and the precomputation storage. For $\sigma \in [2, 32]$, the TwinCDT sampler is at least two times faster than our implementations. However, the TwinCDT sampler requires at least 1.4 KB of memory consumption to store the cumulative dis-

Table 7: Summary of previous works at 3.6 GHz

σ	Number of Samples ($\times 10^6$/sec)	Precomputation Storage (KB)
2 [27]	43.73/53.53/65.51	1.4/4.6/46
8 [27]	32.31/45.69/54.44	3/10/100
32 [27]	29.47/34.08/36.51	9.5/32/318
2^{15} [31]	$\approx$10.59 (online), 1.53 (online+offline)	$2^{5.4}$
$4 - 2^{20}$ [9]	$\approx$13.97	< 1
$1.29 - 1.85$ [20]	8.31	< 1

tribution tables. With the increase of σ, it needs more memory and has worse performance. The performances of our implementations don't change significantly as σ increases, and their memory consumptions are independent of σ.

The convolution sampler [31] is isochronous concerning σ, c, and z, so we compare it with the implementations of Table 6. If we measure the whole running time, our implementations 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 basic samples during the offline phase. In this case, the implementations based on the binary sampler and the CDT sampler generate 8.73×10^6 and 9.16×10^6 samples per second for $\sigma = 2^{15}$, which are slower than the convolution sampler. However, while performing the benchmark test, [31] assumes $\sigma \geq 13 > 4\sqrt{2}\eta_\epsilon(\mathbb{Z})$ for reasonable ϵ, although $\sqrt{2}\eta_\epsilon(\mathbb{Z})$ is the minimum to be usable for the convolution sampler[4]. Under the same assumption, we can set $C(\sigma)$ equal to $(13\lceil k \rceil)/(14k)$. Then our implementations 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 sampling algorithm in [9] is based on the binary sampler and the rejection sampling strategy of Karney's sampler. Although it's faster than our implementations based on the binary sampler, its side-channel resistance perspective is unclear. FALCON's sampler [20] is isochronous concerning σ, c, and z. It's faster than our implementations of Table 6. However, its storage and running time increase rapidly as σ grows like the TwinCDT sampler.

We apply Algorithm 4 to the lattice cryptography library PALISADE [7] and use its latest version as of this writing[5] to evaluate the performances. We replace Karney's sampler in PALISADE with two implementations based on the CDT sampler in Table 5 and Table 6 to sample from the discrete Gaussian distributions over the integers. Karney's sampler doesn't consider the timing attacks. PALISADE uses the BLAKE2 hash function[6] as the PRNG. In PAL-

[4] The smoothing parameter in [31] is $\sqrt{2\pi}$ times that in this paper
[5] https://gitlab.com/palisade/palisade-development/-/tree/release-v1.10.6
[6] https://www.blake2.net

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

ISADE, 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 [19] of IBE [29] and CP-ABE [41]. Both rely on the latest MP trapdoor sampler [15]. 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 based on the CDT sampler in Table 5. 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 Based on 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 our new Gaussian sampler Algorithm 4.

5.1 New COSAC Sampler

The core problem of the COSAC sampler [42] is how to sample from the distributions $D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$ with $c_F \in [-1/2, 1/2]$. It uses the normal distribution $\mathcal{N}(0,\sigma)$ and the rejection sampling to solve this problem. The rejection sampling strategy of the original COSAC sampler is independent of c_F and requires about two trials on average to output a sample. The rejection sampling strategy of our new COSAC sampler changes along with the change of c_F. The expected number of trials of our 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 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 $c_F \in [-1/2, 1/2]$.

Algorithm 7: Rounding Sampler

Input: Standard deviation σ, center $c_F \in [-1/2, 1/2]$
Output: $z \leftarrow D_{\mathbb{Z}\setminus\{0\},\sigma,c_F}$

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

Theorem 8. *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 $P_{true}(\sigma,c)$ of Algorithm 7 satisfies*

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

As we have analyzed the real interval of each nonzero integer, we can prove the correctness of Algorithm 8 and Algorithm 7 by some simple integral calculations. For $P_{true}(\sigma,c)$, we need to precisely calculate the acceptance probability of one trial of Algorithm 7 and estimate the lower bound by Lemma 2. We provide the detailed proof in Appendix D. If σ is far greater than $\eta_\epsilon(\mathbb{Z})$, the acceptance probability $P_{true}(\sigma,c)$ of Algorithm 7 is very close to 1. In theory, Algorithm 8 is almost two times faster than the original COSAC sampler.

Algorithm 8: New COSAC Sampler

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

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

Due to the impossibility of achieving perfect distributions in practice, we can use the following theorem to estimate the security loss of the cryptographic schemes invoking Algorithm 8. The proof of Theorem 9 is similar to that of Theorem 5. For Theorem 9, we need to estimate the relative error of the probability of sampling each nonzero integer by the absolute error $\Delta_{\mathcal{N}}$ of $\mathcal{N}(0,1)$. Because the length of each real interval increases at most $2\sigma\Delta_{\mathcal{N}}$, we can estimate the relative error by some integral calculations. We provide the detailed proof in Appendix E.

Theorem 9. *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 [22] 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 [42]. 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 and the number of trials. The benchmark results are shown in Table 9.

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

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 the original. The main reason is the additional operations to hide c and z. As claimed in [42], our implementation of Algorithm 8 may also reveal σ. Although the implementation involves AVX2 intrinsic instructions, we compare it with the implementations of Algorithm 4 in Table 5. 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 9.

6 Conclusions

In this work, we mainly propose three algorithms about how 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 operations significantly. We apply it to FALCON to reduce 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 apply it to PALISADE and get a more secure and efficient implementation of the MP trapdoor sampler. We can utilize the constant-time implementation of the Knuth-Yao sampler [23] as the basic sampler and improve the performances of our implementations further. Besides, Theorem 7 is only available for search problems. For decision problems, we may prove the security by the techniques in [32].

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.

References

1. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT*, pages 553–572. Springer, 2010.
2. Miklós Ajtai. Generating hard instances of lattice problems. In Gary L. Miller, editor, *STOC*, pages 99–108. ACM, 1996.
3. Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT*, pages 3–24. Springer, 2015.
4. Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. Galactics: Gaussian sampling for lattice-based constant-time implementation of cryptographic signatures, revisited. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *CCS*, pages 2147–2164. ACM, 2019.
5. Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT*, pages 494–524. Springer, 2018.
6. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *STOC*, pages 575–584. ACM, 2013.
7. David Bruce Cousins, Giovanni Di Crescenzo, Kamil Doruk Gür, Kevin King, Yuriy Polyakov, Kurt Rohloff, Gerard W. Ryan, and Erkay Savas. Implementing conjunction obfuscation under entropic ring LWE. In *IEEE Symposium on Security and Privacy*, pages 354–371. IEEE Computer Society, 2018.
8. Wei Dai, Yarkin Doröz, Yuriy Polyakov, Kurt Rohloff, Hadi Sajjadpour, Erkay Savas, and Berk Sunar. Implementation and evaluation of a lattice-based key-policy ABE scheme. *IEEE Trans. Inf. Forensics Secur.*, 13(5):1169–1184, 2018.
9. Yusong Du, Baodian Wei, and Huang Zhang. A rejection sampling algorithm for off-centered discrete Gaussian distributions over the integers. *Sci. China Inf. Sci.*, 62(3):39103:1–39103:3, 2019.
10. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO*, pages 40–56. Springer, 2013.

11. Léo Ducas, Steven D. Galbraith, Thomas Prest, and Yang Yu. Integral matrix gram root and lattice Gaussian sampling without floats. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT*, pages 608–637. Springer, 2020.
12. Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT*, pages 22–41. Springer, 2014.
13. Léo Ducas and Phong Q. Nguyen. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, pages 415–432. Springer, 2012.
14. Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *CCS*, pages 1857–1874. ACM, 2017.
15. Nicholas Genise and Daniele Micciancio. Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT*, pages 174–203. Springer, 2018.
16. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.
17. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Cynthia Dwork, editor, *STOC*, pages 197–206. ACM, 2008.
18. Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload – A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES*, pages 323–345. Springer, 2016.
19. Kamil Doruk Gür, Yuriy Polyakov, Kurt Rohloff, Gerard W. Ryan, Hadi Sajjadpour, and Erkay Savas. Practical applications of improved Gaussian sampling for trapdoor lattices. *IEEE Trans. Computers*, 68(4):570–584, 2019.
20. James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous Gaussian sampling: From inception to implementation. In Jintai Ding and Jean-Pierre Tillich, editors, *PQC*, pages 53–71. Springer, 2020.
21. Yupu Hu and Huiwen Jia. A new Gaussian sampling for trapdoor lattices with arbitrary modulus. *Des. Codes Cryptogr.*, 87(11):2553–2570, 2019.
22. Andreas Hülsing, Tanja Lange, and Kit Smeets. Rounded Gaussians - fast and secure constant-time sampling for lattice-based crypto. In Michel Abdalla and Ricardo Dahab, editors, *PKC*, pages 728–757. Springer, 2018.
23. Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Pushing the speed limit of constant-time discrete gaussian sampling. A case study on the falcon signature scheme. In *DAC*, page 88. ACM, 2019.
24. Charles F. F. Karney. Sampling exactly from the normal distribution. *ACM Trans. Math. Softw.*, 42(1):3:1–3:14, 2016.
25. Vadim Lyubashevsky, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehle, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology (2020), available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`.
26. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT*, pages 1–23. Springer, 2010.
27. Carlos Aguilar Melchor and Thomas Ricosset. CDT-based Gaussian sampling: From multi to double precision. *IEEE Trans. Computers*, 67(11):1610–1621, 2018.

28. Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *FOCS*, pages 356–365. IEEE Computer Society, 2002.
29. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, pages 700–718. Springer, 2012.
30. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *FOCS*, pages 372–381. IEEE Computer Society, 2004.
31. Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO*, pages 455–485. Springer, 2017.
32. Daniele Micciancio and Michael Walter. On the bit security of cryptographic primitives. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT*, pages 3–28. Springer, 2018.
33. Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In Tal Rabin, editor, *CRYPTO*, pages 80–97. Springer, 2010.
34. Chris Peikert. A decade of lattice cryptography. *Found. Trends Theor. Comput. Sci.*, 10(4):283–424, 2016.
35. Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongswan's implementation of post-quantum signatures. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *CCS*, pages 1843–1855. ACM, 2017.
36. Thomas Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT*, pages 347–374. Springer, 2017.
37. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology (2020), available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`.
38. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 84–93. ACM, 2005.
39. Peter Schwabe, Roberto Avanzi, Joppe Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehle. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology (2020), available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions`.
40. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.
41. Jiang Zhang and Zhenfeng Zhang. A ciphertext policy attribute-based encryption scheme without pairings. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Inscrypt*, pages 324–340. Springer, 2011.
42. Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. COSAC: compact and scalable arbitrary-centered discrete Gaussian sampling over integers. In Jintai Ding and Jean-Pierre Tillich, editors, *PQC*, pages 284–303. Springer, 2020.
43. Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. FACCT: fast, compact, and constant-time discrete Gaussian sampler over integers. *IEEE Trans. Computers*, 69(1):126–137, 2020.

A Proof of Theorem 5

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 equal to $2\lambda^{(R)}+1$. 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 5:

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

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

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:

$$\Delta\lambda \leq \log \left(\sqrt{2}^{\frac{a}{a-1}+1} \cdot R_a \left(\mathcal{B}^{(R)}_{p_{rej}}, \mathcal{B}^{(I)}_{p_{rej}} \right)^{\frac{aQ_{exp}}{a-1}} \cdot R_a \left(D^{(R)}_{\mathbb{N},\sigma_0}, D^{(I)}_{\mathbb{N},\sigma_0} \right)^{Q_{bs}} \right). \quad (4)$$

Based on the first condition in Theorem 5 and $a = 2\lambda^{(R)} + 1$, an application of Lemma 7 yields to

$$R_a \left(\mathcal{B}^{(R)}_{p_{rej}}, \mathcal{B}^{(I)}_{p_{rej}} \right) \leq 1 + \frac{a-1}{4aQ_{exp}}. \quad (5)$$

By combining (4) (5) and the second condition in Theorem 5, 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 $x, n > 0$. □

B Proof of Theorem 6

The number of iterations of the while loop in Algorithm 4 follows a geometric distribution of its acceptance probability. When proving Theorem 6, we will need the following lemma [20] to bound the Rényi divergence between two geometric distributions.

Lemma 10 ([20]). *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 6.

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 10 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:

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

□

C Proof of Theorem 7

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 6, 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 5 and Theorem 6 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. □

D Proof of Theorem 8

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 $P_{true}(\sigma, c)$ of Algorithm 7:

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

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

E Proof of Theorem 9

Proof. The proof of Theorem 9 is very similar to that of Theorem 5. 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. $D^{(IN)}_{\mathbb{Z}\setminus\{0\},\sigma,c_f}$) 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 5:

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

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

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 9, 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 7 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. □