Securely Sampling Discrete Gaussian Noise for Multi-Party Differential Privacy

Chengkun Wei*
Zhejiang University
Hangzhou, China
weichengkun@zju.edu.cn

Ruijing Yu*
Zhejiang University
Hangzhou, China
rjyu@zju.edu.cn

Yuan Fan
Zhejiang University
Hangzhou, China
fanyuan@zju.edu.cn

Wenzhi Chen†
Zhejiang University
Hangzhou, China
chenwz@zju.edu.cn

Tianhao Wang†
University of Virginia
Virginia, USA
tianhao@virginia.edu

ABSTRACT

Differential Privacy (DP) is a widely used technique for protecting individuals' privacy by limiting what can be inferred about them from aggregate data. Recently, there have been efforts to implement DP using Secure Multi-Party Computation (MPC) to achieve high utility without the need for a trusted third party. One of the key components of implementing DP in MPC is noise sampling. Our work presents the first MPC solution for sampling discrete Gaussian, a common type of noise used for constructing DP mechanisms, which plays nicely with malicious secure MPC protocols.

Our solution is both generic, supporting various MPC protocols and any number of parties, and efficient, relying primarily on bit operations and avoiding computation with transcendental functions or non-integer arithmetic. Our experiments show that our method can generate 2^{15} discrete Gaussian samples with a standard deviation of 20 and a security parameter of 128 in 1.5 minutes.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; • **Mathematics of computing** → *Probabilistic algorithms*;

KEYWORDS

Differential Privacy; Privacy-Preserving Protocol; Multi-Party Computation

ACM Reference Format:
Chengkun Wei, Ruijing Yu, Yuan Fan, Wenzhi Chen and Tianhao Wang. 2023. Securely Sampling Discrete Gaussian Noise for Multi-Party Differential Privacy. In *Proceedings of Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, Copenhagen, Denmark, November 26–30, 2023 (CCS '23),* 15 pages.
https://doi.org/10.1145/3576915.3616641

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3616641

1 INTRODUCTION

Differential Privacy (DP) is a widely used technique for ensuring that the output of aggregate algorithms does not reveal private details about individuals. This technique is employed by both industry and academia (e.g. Apple [37, 38], Google [4, 17], LinkedIn [34, 35], and Microsoft [11]). A common approach to constructing a DP mechanism is to add random noise to the result of an aggregate function before releasing it. There are several models for implementing DP, including the central model, where a server aggregates data from users and applies noise; the local model, where users add noise to their own data before sending it to a server; and the shuffle model, where a shuffler is added between users and servers to permute and forward the results. However, these solutions either result in a loss of utility (local and shuffle models) or require a trusted third party (central model).

To implement a DP mechanism with high utility while avoiding a trusted party, several works [5, 6, 33] leverage the technique of Secure Multi-Party Computation (MPC). MPC is a cryptographic tool that allows two or more parties to jointly compute an agreed-upon function without revealing any private information other than the output. By using MPC, users can simulate the central model algorithm without a trusted server, and each user only learns their own input and the differentially private result.

Noise sampling is an important aspect of implementing DP mechanisms in MPC. The noise commonly used in DP includes Laplace and Gaussian noise (both continuous and discrete). Compared to Laplace noise, the Gaussian noise is better suited for applications with a high degree of composition (i.e. answering many queries with independent noise) due to its lighter tails [7]. And compared to continuous Gaussian noise, the discrete Gaussian noise can be naturally represented on finite computers, avoiding tricking issues caused by the approximation of continuous noise [7].

Several works [6, 21, 22, 24] propose MPC protocols for noise sampling based on *distributed noise generation*, where each party provides a partial of noise which is then securely combined. Such a solution is efficient but only provides semi-honest security. Other works propose sampling methods for continuous Laplace [1, 16], discrete Laplace [33], and continuous Gaussian [8, 15] which can be easily converted to malicious secure MPC protocols. To the best of

*Chengkun and Ruijing are co-first authors.
†Corresponding authors.

our knowledge, there is no solution for sampling discrete Gaussian that plays nicely with malicious secure MPC protocols, and our work fills this gap.

We design an MPC solution for discrete Gaussian sampling based on rejection sampling. First, we ask each party to provide uniform bits as input. By XORing the input from each party, we can obtain correct uniform bits as long as one party provides the actual randomness. Our method consists of two parts: sampling the proposal distribution and determining whether to accept the sample. We use discrete Laplace as the proposal distribution and include a circuit for accepting or rejecting a sample. Our solution performs efficiently when the standard deviation σ is large ($\sigma \geq 5$). And we found that the basic method works better when σ is smaller. Our experiments show that our method can generate 2^{15} discrete Gaussian samples with $\sigma = 20$ and a security parameter of 128 in 1.5 minutes.

The remainder of this paper is organized as follows: In Section 2, we describe the preliminary of our method. In Section 3, we refine our problem. We present our sampling method in detail in Section 4, and analyze the privacy and complexity of our method in Section 5 and 6. We provide the evaluation in Section 7, describe related work in Section 8 and conclude in Section 9.

2 PRELIMINARY

2.1 Secure Multiparty Computation

Secure multiparty computation (MPC) [18, 39] allows two or more parties $P_1, ..., P_n$ to jointly compute an agreed-upon function $y = f(x_1, ..., x_n)$, where x_i is the private input provided by P_i. MPC guarantees that each P_i learns nothing except the result y and its own input x_i. Specifically, each P_i cannot learn the input $x_j, j \neq i$ of other parties, nor any intermediate value derived from x_j. Secure two-party computation (2PC) is a special type of MPC in which only two parties P_1, P_2 are involved in the computation.

To implement MPC for a given function f, a common method is to convert f into a *circuit* $\mathcal{C}$ and then apply a generic MPC protocol. The circuit is a model of computation in which input values proceed through a sequence of gates, each of which computes a basic function. Boolean circuits and arithmetic circuits are two different types of circuits, where the gates of Boolean circuits correspond to bit operations (e.g., XOR, AND), and the gates of arithmetic circuits correspond to arithmetic operations (e.g., addition, multiplication). Once a circuit is designed, it can be applied to various types of generic MPC protocols.

Two main types of MPC protocols are *garbled circuit* based protocols (e.g., Yao [39]) and *secret sharing* based protocols (e.g., GMW [20]). The garbled circuit-based protocols include two characters, the *garbler* and *evaluator*. The garbler encrypts the circuit $\mathcal{C}$ and all inputs x to garbled form $[[\mathcal{C}]]$, $[[x]]$ via some **Garble** function, then sends them to the evaluator. The evaluator evaluates the garbled circuits and obtains the garbled output $[[y]]$. Then they reveal the garbled output via some **Reveal** function to get the plain output y. In secret sharing-based protocols, each party "shares" its input x_i to n pieces $\langle x_i \rangle_1, \langle x_i \rangle_2, ..., \langle x_i \rangle_n$ (corresponding to n parties) by some **Share** function. Then, each party j evaluates the circuit and obtains the share $\langle y \rangle_j$ of the output, which can be combined to plain output y via some **Rec** function, i.e. $\mathbf{Rec}(\langle y \rangle_1, \langle y \rangle_2, ..., \langle y \rangle_n) = y$. Generally, the garbled circuit-based protocol is suitable for the Boolean circuit, and the secret-share-based is suitable for the arithmetic circuit. The adversary models supported by MPC protocols typically include the *semi-honest* model, where adversaries are curious about privacy but follow the protocol; and the *malicious* model, where the adversaries may violate the protocol.

2.2 Differential Privacy

Differential privacy is concerned with providing aggregate information about a dataset $D \in \mathbb{D}^*$ without disclosing information about specific individuals. DP captures the notion that for two neighboring datasets, their aggregate result distributions should be "similar" when applying the DP mechanism.

Definition 2.1 (Differential Privacy). Let $\epsilon, \delta > 0$. A randomized algorithm $\mathbf{A} : \mathbb{D}^* \to \mathbb{O}$ satisfies (ϵ, δ)-DP, if for any neighbouring $D, D' \in \mathbb{D}^*$, and $\mathbb{S} \subset \mathbb{O}$:

$$\Pr[\mathbf{A}(D) \in \mathbb{S}] \leq e^{\epsilon} \Pr\left[(\mathbf{A}(D') \in \mathbb{S}\right] + \delta$$

where, the parameter ϵ represents the privacy budget, and δ can be roughly understood as the failure probability (fail to satisfy ϵ-DP). The smaller the ϵ and δ, the stronger the privacy guarantees.

The main technique to construct DP mechanism $\mathbf{A}$ is to add noise $\mathbf{r}$ to the output of the aggregation function q, s.t.

$$\mathbf{A}(x_1, x_2, ..., x_n) = q(x_1, x_2, ..., x_n) + \mathbf{r}$$

where x_i is the raw data from user i. And $\mathbf{r}$ is noise commonly sampled from Laplace and Gaussian distributions.

DP Implementation Model. There exist different models for implementing DP mechanisms. A natural DP implementation, called the *central model*, assumes a trusted central server collects raw data from each user. The server aggregates the raw data, adds noise, and publishes the differentially private result, which achieves optimal utility. In the industry setting, there is a need to get rid of the trust in the central server because individuals do not trust companies as they do government agencies. Thus when companies collect data, another implementation called the *local model* [28] is used. The local model assumes an untrusted server and each client locally adds noise to its raw data before sending it to the server for aggregation. However, the local model suffers from poor utility because noise is added multiple times to the result [28, 30]. The *shuffle model* [4] is an intermediate between the central and local model, in which a trusted shuffler (which does not collude with any party) is added between clients and an untrusted server in the local model. The shuffler permutes and forwards the clients' randomized result. The permutation breaks the mapping between a client and its corresponding result, which reduces the noise requirement. While the shuffle model has a higher utility than the local model, it is weaker than the central model [2, 10] and still needs a trusted party.

2.3 Achieving Differential Privacy with MPC

To implement a DP mechanism with high utility while avoiding a trusted party, some works leverage the MPC technique [5, 6, 33]. MPC allows clients to securely execute the central model algorithm without involving a central server, in which each client only learns its corresponding raw data and the differentially private results. After finishing the MPC, the clients can directly send the differentially private results to the untrusted target server for publication.

When the number of clients n is large, performing n-party MPC to implement DP is complicated and expensive. A practical way to implement a DP mechanism in MPC is to assume two untrusted but non-colluding servers, a target server (for publishing differentially private results), and an auxiliary server. At first, the clients secret-share their raw data and send it to two servers, which then run a 2PC for aggregation, noise sampling, and addition. In particular, the whole process is

(1) *Secret Sharing.* Each client i secret-shares its raw data x_i to two encrypted values $\langle x_i \rangle_1, \langle x_i \rangle_2 = \mathbf{Share}(x_i)$, then sends $\langle x_i \rangle_1$ to the target server and $\langle x_i \rangle_2$ to the auxiliary server.
(2) *Aggregation.* The target and auxiliary servers run a secret sharing-based 2PC to compute the aggregation function q based on the shares sent by the clients. After 2PC, two servers get the shares $\langle t \rangle_1, \langle t \rangle_2$ of the aggregation result respectively.
(3) *Noise Sampling and Addition.* The target and auxiliary servers runs a 2PC to compute $f(\langle t \rangle_1, \langle t \rangle_2) = \mathbf{Rec}(\langle t \rangle_1, \langle t \rangle_2) + \mathbf{Noise}()$ and get the final differentially private result, where **Noise** is sampling process to generate the noise **r** in DP mechanism.

2.4 Discrete Gaussian on Differential Privacy

Challenges of Gaussian Noise on Differential Privacy. Though Gaussian noise is widely applied to construct the DP mechanism, it has a continuous form that presents several practical challenges. First, finite computers cannot exactly represent the continuous noise, and the naïve use of finite-precision approximations can result in catastrophic failures of privacy [23, 31]. Second, when the aggregation data is discrete, adding continuous noise makes the result less interpretable. Third, the sampling of Gaussian noise (e.g. Box-Muller method) may include expensive high-precision floating point arithmetic and computation on transcendental functions.

To avoid this problem, Canonne, Kamath, and Steinke [7] propose to use *discrete* Gaussian noise to construct the DP mechanism. The discrete Gaussian distribution is a natural discrete analog of the Gaussian distribution, and it is shown that adding discrete Gaussian noise provides essentially the same privacy and accuracy guarantees as the addition of continuous Gaussian noise [7].

Definition 2.2 (Discrete Gaussian Distribution). For parameter $\sigma > 0$, the discrete Gaussian distribution $\mathcal{N}_{\mathbb{Z}}(\sigma)$ is define as

$$f_{\mathcal{N}_{\mathbb{Z}}}(x;\sigma) = c \cdot e^{-\frac{x^2}{2\sigma^2}} \text{ for } x \in \mathbb{Z}$$

where c is the normalized constant.

Still, the discrete Gaussian distribution has an infinite range $\mathbb{Z}$ and we cannot necessarily represent all samples in finite computers. For implementation, we can instead sample from a truncated distribution with finite range $(-N, N) \cap \mathbb{Z}$ for some $N \in \mathbb{N}^+$. The truncated discrete Gaussian distribution $\mathcal{N}_{\mathbb{Z},N}(\sigma)$ is defined as

$$f_{\mathcal{N}_{\mathbb{Z},N}}(x;\sigma) = \frac{f_{\mathcal{N}_{\mathbb{Z}}}(x;\sigma)}{\sum_{|t|<N} f_{\mathcal{N}_{\mathbb{Z}}}(t;\sigma)} \quad \forall x \in (-N, N) \cap \mathbb{Z}$$

The statistical distance between $\mathcal{N}_{\mathbb{Z}}(\sigma)$ and $\mathcal{N}_{\mathbb{Z},N}(\sigma)$ is bounded by $2e^{-\frac{N^2}{2\sigma^2}}$, which can be incorporated into δ in DP [7].

Table 1: Comparison with previous work that samples noises in MPC.

Distribution	Security	Work
Continuous Laplace	Semi-honest	[6, 21]
	Malicious	[1, 16]
Discrete Laplace	Semi-honest	[21]
	Malicious	[8, 15]
Continuous Gaussian	Semi-honest	[22]
	Malicious	[33]
Discrete Gaussian	Semi-honest	[24]
	Malicious	Ours

3 SAMPLING DISCRETE GAUSSIAN IN MPC

3.1 System Model

We assume there are two non-colluding servers, a target server (e.g., Census Bureau) and an auxiliary server (e.g., one from a university), who want to jointly generate the discrete Gaussian noise for DP implementation. Neither party can reveal the noise-free result, the target server will get the differentially private result and can choose to share/publish it. Specifically, these two servers run a 2PC protocol to compute the sampling function $r = \mathbf{Noise}(r_1, r_2)$, where r is the discrete Gaussian noise, r_1 and r_2 are randomized inputs from the two parties, in order to provide the randomness.

We consider a security model where parties are non-colluding and reasonable. That is, the target server will be malicious only for inferring private information (not for blocking the execution or poisoning the result, since it would expect correct noise for subsequent publication), while the auxiliary party is interested in inferring or poisoning the result. As to the clients, there is nothing preventing a client from lying about their true value, when they secretly share their raw data. The best we can do is to ensure (through e.g., zero-knowledge proofs) that the shares are valid after they are combined.

3.2 Existing Solutions

Several work [6, 21, 22, 24] suggested using distributed noise generation in MPC for sampling noises, where each party locally computes partial noises, which are securely combined. Distributed noise generation is efficient but provides only semi-honest security since each party must provide correct partial noise. There exist some sampling methods that can be easily converted to malicious secure MPC protocols, but they focus on sampling continuous Laplace [1, 16], continuous Gaussian [33] and discrete Laplace [8, 15]. Notice that though our security model is not fully malicious, our solution can be easily applied to support fully malicious security (see Section 4.7). We compare the existing solutions with our solution in terms of noise type and supported security model in Table 1.

4 OUR METHOD

An efficient way to avoid poisoning attack is to require each party to provide uniform random bits as input. By XORing the input from each party, one can get the correct uniform bits as long as one of the parties provides the actual randomness. This method does not require an additional check.

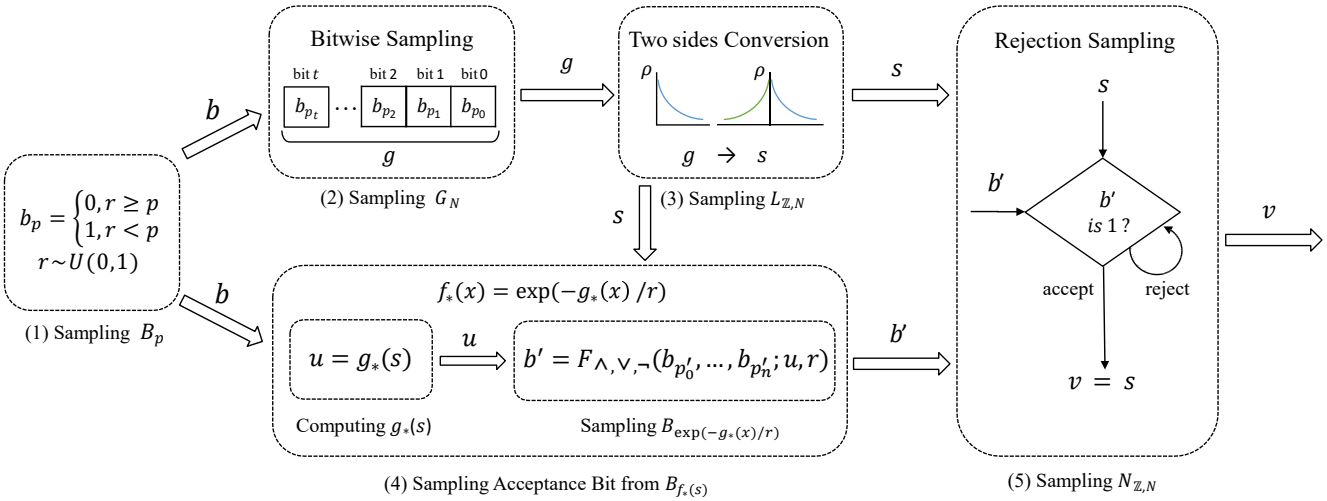

Figure 1: Structure of our sampling method. (1) Sampling b from the Bernoulli distribution $\mathcal{B}$; (2) Bitwise sampling of g from the geometric distribution $\mathcal{G}_N$, which independently samples each bit i of the geometric sample from a specified Bernoulli distribution $\mathcal{B}(p_i)$ (Section 4.3.1); (3) Convert g into s, which follows discrete Laplace distribution $\mathcal{L}_{\mathbb{Z},N}$ (Section 4.3.2); (4) Sampling the acceptance bits from Bernoulli distribution $\mathcal{B}(f_*(s))$, which combines by the computing $u = g_*(s)$ and the sampling of Bernoulli distribution $\mathcal{B}(\exp(-u/r))$. The sampling of $\mathcal{B}(\exp(-u/r))$ combines several Bernoulli samples by some bit operations (more details in Section 4.4); (5) Rejection sampling of discrete Gaussian distribution $\mathcal{N}_{\mathbb{Z},N}$, with discrete Laplace as proposal distribution.

We construct our 2PC protocol for sampling based on this XOR technique. This requires us to design a sampling function f based on uniform bits (as ordinary sampling). Then, we can apply a generic 2PC protocol to compute the function $\mathbf{Noise}(r_1, r_2) = f(r_1 \oplus r_2)$ where r_1, r_2 are the uniform bits provided by two parties respectively. A semi-honest generic 2PC protocol (e.g., Yao [39]) can prevent parties from inferring private information. And it's sufficient to satisfy our security model by applying a semi-honest protocol and XOR technique. Besides, our solution can be extended to satisfy fully malicious security as described in Section 4.7.

4.1 Overview

We construct our sampling method for (truncated) discrete Gaussian based on rejection sampling. Its idea is to avoid directly sampling from a complicated distribution by first sampling from an easier distribution and then rejecting the samples to make the resulting accepted samples follow the target distribution.

More specifically, denote $\mathcal{T}$ as the target distribution and $\mathcal{P}$ as the proposal distribution. First, the rejection sampling chooses the smallest M such that $\forall x\ M \cdot f_{\mathcal{P}}(x) \geq f_{\mathcal{T}}(x)$. Then the sample phase is to first sample s from $\mathcal{P}$, then sample bit b from Bernoulli distribution $\mathcal{B}(f_*(s))$ to decide whether to accept the sample, where $f_*(x) = \frac{f_{\mathcal{T}}(x)}{M \cdot f_{\mathcal{P}}(x)}$ (i.e., accept s with probability $\frac{f_{\mathcal{T}}(s)}{M \cdot f_{\mathcal{P}}(s)}$).

For the proposal distribution, we choose the (truncated) discrete Laplace distribution as the proposal distribution since it is close to the target discrete Gaussian distribution (with average acceptance probability > 0.29 [7]), and has an efficient bitwise sampling circuit [15]. For sampling the acceptance bit, i.e., sampling from $\mathcal{B}(f_*(s))$, we observe that $f_*(x)$ includes the exponential computation such that $f_*(x) = \exp(-g_*(x)/r)$ where $g_*(x) \geq 0$ is a polynomial and $r \in \mathbb{R}^+$ (see Section 4.4). To avoid expensive exponential computation, inspired by Ducas et al. [12], we first build a basic sampling circuit for Bernoulli distribution $\mathcal{B}(\exp(-u/r))$ where u, r are parameters which require that $u \geq 0$ is an *integer* and $r \in \mathbb{R}^+$. This circuit only applies some bit operations on several Bernoulli samples without exponential computation. Then, we carefully choose the discrete Laplace parameter to ensure that $g_*(x)$ returns an integer. Finally, the sampling of the acceptance bit can be done by first computing $u = g_*(s)$ and then sampling from $\mathcal{B}(\exp(-u/r))$.

To implement rejection sampling in secure computation, we first generate a fixed number of m samples from a proposal distribution and their corresponding acceptance bits, and then reveal all the acceptance bits in public and adopt the first accepted sample. Revealing the acceptance bits does not leak the privacy of the accepted sample since the number of reject times is independent of the final accepted sample [8]. Such an approach can be extended to generate n target samples by adopting the first n accepted samples. Besides, one should generate enough proposal samples to make the probability of not getting enough accepted target samples less than a certain upper bound (depending on the specific setting).

Overall, our sampling method mainly includes bit operations (and a few integer arithmetics), without including non-integer arithmetic (e.g. floating-point arithmetic), and non-linear computation (e.g. exponential, logarithmic). Therefore, we implement the circuits in our method in the form of Boolean circuits. Our work is a generic solution since our sampling circuit can be applied to different generic 2PC protocols (and our solution can also be easily extended to MPC, see Section 4.7). Figure 1 shows the structure of our sampling method for discrete Gaussian. In the following section, we will describe each part in detail.

Algorithm 1: SamplingCircuit-$\mathcal{B}(p)$ [15]

Input: Parameter $p = 0.b_0...b_{\mu-1}$ in μ-bits binary form
Output: A sample x from $\mathcal{B}(p)$

$x \leftarrow \textbf{BIT}(1)$
for $i \leftarrow \mu - 1$ **to** 0 **do**
  $r \leftarrow \textbf{URBIT}()$ ▷ a uniform random bit
  $t \leftarrow \textbf{XOR}(b_i, r)$
  $x \leftarrow \textbf{MUX}(t, \textbf{NOT}(r), x)$ ▷ returns $\neg r$ if $t = 1$ else x
end
return x

Algorithm 2: SamplingCircuit-$\mathcal{G}_N(p)$

Input: Parameter p, range parameter $N = 2^\kappa$
Output: A sample x from $\mathcal{G}_N(p)$
Precompute: $p_i = p^{2^i}/(1 + p^{2^i})$ for $i = 0, ..., \kappa - 1$

for $i \leftarrow 0$ **to** $\kappa - 1$ **do**
  $b_i \leftarrow$ **SamplingCircuit-**$\mathcal{B}(p_i)$
end
return $\textbf{COMPOSE}(b_0, b_1, ..., b_{\kappa-1})$ ▷ returns $x = \sum_{i=0}^{\kappa-1} 2^i b_i$

4.2 Sampling Bernoulli Distribution

The sampling circuit of Bernoulli distribution $\mathcal{B}(p)$ is a basic building block in our method. In particular, the sampling of $\mathcal{B}(p)$ for a given $p \in [0, 1]$ is to first generate $r \in [0, 1]$ uniformly then output 1 if $r < p$ and 0 otherwise. A direct circuit implementation proposed by Dwork et al. [15] is to approximately represent p and r in μ-bits binary form, first sample r by using μ uniform bits, then perform a comparison between p and r. Due to the approximation, the actual distribution of the output has at most $2^{-\mu}$ statistical distance between the true distribution.

4.3 Sampling Discrete Laplace Distribution

The discrete Laplace distribution $\mathcal{L}_\mathbb{Z}(t)$ is defined as

$$f_\mathcal{L}(x; t) = c \cdot e^{-|x|/t} \text{ for } x \in \mathbb{Z}$$

where c is the normalizing constant.

Generally, one can sample discrete Laplace based on *inverse transform sampling*. The sampling is to first generate $u \in [0, 1]$ uniformly, then compute $F^{-1}(u)$, where F is the cumulative distribution function (CDF). However, the inverted CDF of discrete Laplace includes an expensive logarithm computation $\ln x$.

To generate a discrete Laplace sample while avoiding expensive logarithm computation, we first apply the *bitwise* method introduced by Dwork et al. [15] to generate a geometric sample, then convert it to a two-sided discrete Laplace sample.

4.3.1 Sampling Geometric Distribution. The geometric distribution $\mathcal{G}(p)$ is defined as

$$f_\mathcal{G}(x; p) = (1 - p) \cdot p^x \text{ for } x \in \mathbb{N}$$

Dwork et al. [15] observed that the geometric sample has a special structure that each bit in its binary representation is independent. Therefore, the generation of a geometric sample can be reduced to independently generating a Bernoulli sample for each bit. By generating κ bits and merging them, one can obtain a geometric sample with truncated range $[0, 2^\kappa)$ (i.e. $\mathcal{G}_{2^\kappa}(p)$). In particular, the bit i in the geometric sample from $\mathcal{G}(p)$ follows Bernoulli distribution $\mathcal{B}(p^{2^i}/(1 + p^{2^i}))$ [15]. Therefore, one can construct sampling circuit of $\mathcal{G}_{2^\kappa}(p)$ by merging the output of κ Bernoulli sampling circuits that sample from $\mathcal{B}(p^{2^i}/(1 + p^{2^i}))$ for bit $i = 0, ..., \kappa - 1$, as shown in Algorithm 2.

4.3.2 Converting Geometric to Discrete Laplace. Once we obtain a geometric sample, we can convert it to a two-sided discrete Laplace sample. Notice that the geomtric distribution $\mathcal{G}(e^{-1/t})$ is the "positive half" of the discrete Laplace distribution $\mathcal{L}_\mathbb{Z}(t)$. Therefore, to obtain a $\mathcal{L}_\mathbb{Z}(t)$ sample with range $(-N, N)$ (i.e. $\mathcal{L}_{\mathbb{Z},N}(t)$) the folklore conversion method is to:

(1) Sample x from $\mathcal{G}_N(e^{-1/t})$
(2) Sample a uniform bit s to decide the sign of x.
(3) If $x \neq 0$, output sx. If $x = 0$, and if $s = 0$, output x, else, resample x (back to step 1).

Step 2 converts a geometric sample in range $[0, N)$ to a sample in range $(-N, 0]$ or $[0, N)$, with equal probability. And step 3 needs to perform resampling since the probability of 0 is double-counted in both ranges $(-N, 0]$ and $[0, N)$. This method is exactly a rejection sampling method since it resamples with 50% probability when the geometric sample is 0. In particular, the acceptance probability is

$$p_* = 1 - \frac{1}{2} \cdot f_{\mathcal{G}_N}(0; e^{-1/t}) \in (\frac{1}{2}, 1)$$

Therefore, to generate a discrete Laplace sample, one needs to consume $1/p_* \in (1, 2)$ geometric samples on average.

We propose an efficient conversion method without resampling, which costs only one geometric sample for each discrete Laplace sample. Our starting point is to avoid double-counting in conversion. In particular, if we are able to generate a geometric sample x in the range $[1, N)$, then we can convert it to a two-sided sample y with range $(-N, -1] \cup [1, N)$ according to a uniform bit. Since the range $[1, N)$ and $(-N, -1]$ are disjoint, this process includes no double-counting and there is no need for resampling. After we obtain y, we can generate a discrete Laplace sample z in the range $(-N, N)$, by deciding whether to output 0 or y based on their probabilities.

Next, we consider how to generate a geometric sample in the range $[1, N)$. Assume that we have a sample v from geometric distribution $\mathcal{G}(p)$ with range $[a, b]$ $(a, b \in \mathbb{N})$, s.t.

$$\Pr[v = x] = c_1 \cdot p^x \text{ for } x \text{ in } [a, b] \cap \mathbb{N}$$

Here c_1 is the normalizing constant. We observe that by adding $t \in \mathbb{N}^+$ to v, we obtain a sample $v' = v + t$, s.t.

$$\begin{aligned}\Pr[v' = x] &= \Pr[v = x - t] \\ &= c_1 \cdot p^{x-t} = c_2 \cdot p^x \text{ for } x \text{ in } [a + t, b + t] \cap \mathbb{N}\end{aligned}$$

Here c_2 is the normalizing constant. We can see that v' follows the geometric distribution with range $[a + t, b + t]$. Therefore, to generate a geometric sample x in range $[1, N)$, we can first generate a geometric sample x' in range $[0, N - 1)$ (i.e. $\mathcal{G}_{N-1}$), then obtain $x = x' + 1$. Here x' can be sampled by the bitwise method if $N - 1 = 2^\kappa$. The correctness of our conversion is shown in Theorem 4.1.

Algorithm 3 shows our sampling circuit of $\mathcal{L}_{\mathbb{Z},N}(t)$ with $N = 2^\kappa + 1$, which costs exactly one geometric sample to generate a

Algorithm 3: SamplingCircuit-$\mathcal{L}_{\mathbb{Z},N}(t)$

Input: Parameter t, range parameter $N = 2^\kappa + 1$
Output: A sample z from $\mathcal{L}_{\mathbb{Z},N}(t)$
Precompute: $p = e^{-1/t}, p_0 = f_{\mathcal{L}_{\mathbb{Z},N}}(0;t)$

$b \leftarrow$ **SamplingCircuit-**$\mathcal{B}(p_0)$
$x \leftarrow$ **SamplingCircuit-**$\mathcal{G}_{N-1}(p)$
$s \leftarrow$ **URBIT**() ▷ a uniform random bit
$v \leftarrow$ **MUX**$(b, 0,$ **ADD**$(x, 1))$ ▷ $v = 0$ if $b = 1$ else $x + 1$
return **MUX**$(s, v,$ **NEG**$(v))$ ▷ returns v if $s = 1$ else $-v$

discrete Laplace sample without the resampling process. Overall, it costs $\kappa + 1 = O(\log N)$ Bernoulli samples to generate a discrete Laplace sample with range $(-N, N)$.

Theorem 4.1. *Let $t > 0, N \in \mathbb{N}^+$ with $N \geq 2, x \sim \mathcal{G}_{N-1}(e^{-1/t}), s$ is a uniform bit. Let*

$$z = \begin{cases} 0 & \text{with probability } f_{\mathcal{L}_{\mathbb{Z},N}}(0;t) \\ (2s-1)(x+1) & \text{otherwise} \end{cases}$$

then $z \sim \mathcal{L}_{\mathbb{Z},N}(t)$.

Proof: First, we have $\Pr[z = 0] = f_{\mathcal{L}_{\mathbb{Z},N}}(0;t)$

Then, for any integer value $w \neq 0$, we have

$$\begin{aligned} \Pr[z = w] &= (1 - f_{\mathcal{L}_{\mathbb{Z},N}}(0;t)) \cdot \frac{1}{2} \cdot f_{\mathcal{G}_{N-1}}(|w| - 1; e^{-1/t}) \\ &= (1 - f_{\mathcal{L}_{\mathbb{Z},N}}(0;t)) \cdot \frac{1}{2} \cdot c_3 \cdot e^{-(|w|-1)/t} \\ &= (1 - f_{\mathcal{L}_{\mathbb{Z},N}}(0;t)) \cdot \frac{1}{2} \cdot c_3 \cdot e^{1/t} \cdot e^{-|w|/t} \end{aligned}$$

Here $\frac{1}{2}$ corresponds to the probability of uniform bit s being 0, c_3 is the normalizing constant of $\mathcal{G}_{N-1}(e^{-1/t})$. Let c_4 be the constant that

$$c_4 = (1 - f_{\mathcal{L}_{\mathbb{Z},N}}(0;t)) \cdot \frac{1}{2} \cdot c_3 \cdot e^{1/t}$$

We can get

$$\Pr[z = w] = c_4 \cdot e^{-|w|/t} = f_{\mathcal{L}_{\mathbb{Z},N}}(w;t)$$

Here c_4 can be seen as the normalizing constant of $\mathcal{L}_{\mathbb{Z},N}(t)$.

Therefore, we have $\Pr[z = w] = f_{\mathcal{L}_{\mathbb{Z},N}}(w;t)$ for all $w \in \mathbb{Z}$, which means that $z \sim \mathcal{L}_{\mathbb{Z},N}(t)$.

4.4 Sampling Acceptance Bit

Recall that the acceptance bit is sampled from the Bernoulli distribution $\mathcal{B}(f_*(s))$, where s is a discrete Laplace sample, $f_*(x) = f_{\mathcal{N}_{\mathbb{Z},N}}(x;\sigma)/(M \cdot f_{\mathcal{L}_{\mathbb{Z},N}}(x;t))$, M is the smallest constant s.t. $\forall x\ M \cdot f_{\mathcal{L}_{\mathbb{Z},N}}(x;\sigma) \geq f_{\mathcal{N}_{\mathbb{Z},N}}(x;t)$.

Here we give the expression of $f_*(x)$. The constant M is

$$\begin{aligned} M &= \sup_x \frac{f_{\mathcal{N}_{\mathbb{Z},N}}(x;\sigma)}{f_{\mathcal{L}_{\mathbb{Z},N}}(x;t)} = \sup_x \frac{c_1 \cdot e^{-x^2/(2\sigma^2)}}{c_2 \cdot e^{-|x|/t}} \\ &= \sup_x \frac{c_1}{c_2} \cdot \exp\left(\frac{-(|x| - \sigma^2/t)^2}{2\sigma^2} + \frac{\sigma^2}{2t^2}\right) = \frac{c_1}{c_2} \cdot e^{\sigma^2/(2t^2)} \end{aligned}$$

Here c_1 and c_2 are the normalizing constants of $\mathcal{N}_{\mathbb{Z},N}(\sigma)$ and $\mathcal{L}_{\mathbb{Z},N}(t)$ respectively, such that

$$c_1 = \frac{1}{\sum_{|x|<N} e^{-x^2/(2\sigma^2)}} \quad c_2 = \frac{1}{\sum_{|x|<N} e^{-|x|/t}}$$

Therefore $f_*(x)$ becomes

$$\begin{aligned} f_*(x) &= \frac{f_{\mathcal{N}_{\mathbb{Z},N}}(x;\sigma)}{M \cdot f_{\mathcal{L}_{\mathbb{Z},N}}(x;t)} = \frac{1}{M} \cdot \frac{c_1 \cdot e^{-x^2/(2\sigma^2)}}{c_2 \cdot e^{-|x|/t}} \\ &= \frac{1}{M} \cdot \frac{c_1}{c_2} \cdot \exp\left(\frac{-(|x| - \sigma^2/t)^2}{2\sigma^2} + \frac{\sigma^2}{2t^2}\right) \\ &= \frac{c_2}{c_1} \cdot e^{-\sigma^2/(2t^2)} \cdot \frac{c_1}{c_2} \cdot \exp\left(\frac{-(|x| - \sigma^2/t)^2}{2\sigma^2} + \frac{\sigma^2}{2t^2}\right) \\ &= \exp\left(-\frac{(|x| - \sigma^2/t)^2}{2\sigma^2}\right) \end{aligned}$$

The general method for sampling $\mathcal{B}(f_*(s))$ is to first compute $v = f_*(s)$, then sample from $\mathcal{B}(v)$ by using the Bernoulli sampling method in Section 4.2. However, it involves evaluating exponential functions when computing $v = f_*(s)$.

To build an efficient sampling circuit for the acceptance bit without including exponential computation, we first build a sampling circuit for $\mathcal{B}(\exp(-u/r))$, where u is an integer in $[0, 2^l)$ and $r \in \mathbb{R}^+$. The idea is that the appropriate combination of Bernoulli samples can produce a new Bernoulli sample with a combined parameter [12]. Typically, if one has access to Bernoulli samples $x \sim \mathcal{B}(a)$ and $y \sim \mathcal{B}(b)$, then the Bernoulli sample $z \sim \mathcal{B}(ab)$ can be obtained by $z = x \wedge y$. Notice that $\mathcal{B}(\exp(-u/r))$ can be written as

$$\mathcal{B}\big(\exp(-u/r)\big) = \mathcal{B}\left(\exp(-\sum_{i=0}^{l-1} u_i 2^i/r)\right) = \mathcal{B}\left(\prod_{i=0}^{l-1} \exp(-u_i 2^i/r)\right)$$

where $u = \sum_{i=0}^{l-1} u_i 2^i$ is the binary form of u with $u_i \in \{0, 1\}$. Assume that $b_i \sim \mathcal{B}(\exp(-u_i 2^i/r))$, similarly, the Bernoulli samples $b \sim \mathcal{B}(\exp(-u/r))$ can be obtained by

$$b = b_0 \wedge b_1 \wedge b_2 \wedge \ldots \wedge b_{l-1} = \bigwedge_{i=0}^{l-1} b_i$$

Notice that when $u_i = 1$, $b_i \sim \mathcal{B}(\exp(-2^i/r))$ and when $u_i = 0$, $b_i \sim \mathcal{B}(1)$ so b_i is always 1. Assume that $b'_i \sim \mathcal{B}(\exp(-2^i/r))$, then x can be represented by

$$b = \bigwedge_{u_i=1} b'_i = \bigwedge_{i=0}^{l-1} \neg u_i \vee b'_i$$

Therefore, to sample b from $\mathcal{B}(\exp(-u/r))$, one can first sample b'_i from $\mathcal{B}(\exp(-2^i/r))$ then output $b = \bigwedge_{i=0}^{l-1} \neg u_i \vee b'_i$. The sampling circuit of $\mathcal{B}(\exp(-u/r))$ is shown in Algorithm 4.

After building the sampling circuit for $\mathcal{B}(\exp(-u/r))$, we can apply it to sample $\mathcal{B}(f_*(s))$. As pre-computation, we represent

$$f_*(x) = \exp\left(-\frac{(|x| - \sigma^2/t)^2}{2\sigma^2}\right) = \exp\left(-\frac{g_*(x)}{r}\right)$$

where $g_*(x) \geq 0$ is an integer function (in order to satisfy $u = g_*(s) \in \mathbb{N}$) and $r \in \mathbb{R}^+$. The sampling process for $\mathcal{B}(f_*(s))$ is to first compute $u = g_*(s)$ then apply above method to sample from $\mathcal{B}(\exp(-u/r))$. We will show how to select the discrete Laplace parameter t to ensure $f_*(x) = \exp(-g_*(x)/r)$ in the next section.

Algorithm 4: SamplingCircuit-$\mathcal{B}_{\mathbf{exp}}(u, r)$

Input: Integer $u = \sum_{i=0}^{l-1} u_i 2^i$ with $u_i \in \{0, 1\}$, $r > 0$
Output: A sample b from $\mathcal{B}(\exp(-u/r))$
Precompute: $p_i = \exp(-2^i/r)$ for $i = 0, ..., l-1$

$b \leftarrow \mathbf{BIT}(1)$
for $i \leftarrow 0$ **to** $l-1$ **do**
  $b'_i \leftarrow$ **SamplingCircuit-**$\mathcal{B}(p_i)$
  $v \leftarrow \mathbf{OR}(\mathbf{NOT}(u_i), b'_i)$
  $b \leftarrow \mathbf{AND}(b, v)$
end
return b ▷ $b = \bigwedge_{i=0}^{l-1} \neg u_i \vee b'_i$

4.5 Selection of Discrete Laplace Parameter t

In this section, we discuss our selection of the discrete Laplace parameter t for a given discrete Gaussian distribution $\mathcal{N}_{\mathbb{Z},N}(\sigma)$.

First and foremost, we would like to choose parameter t to achieve a high acceptance probability p_*. Notice that in rejection sampling, the acceptance probability p_* becomes

$$p_* = \sum_s f_{\mathcal{P}}(s) \cdot \frac{f_{\mathcal{T}}(s)}{M \cdot f_{\mathcal{P}}(s)} = \frac{1}{M} \sum_s f_{\mathcal{T}}(s) = \frac{1}{M}.$$

where $\mathcal{P}$ is the proposal distributon and $\mathcal{T}$ is the target distribution.

According to Section 4.4, we have

$$p_* = \frac{1}{M} = \frac{\sum_{|x|<N} e^{-x^2/(2\sigma^2)}}{\sum_{|x|<N} e^{-|x|/t}} \cdot e^{-\sigma^2/(2t^2)}$$

The direct idea of choosing t is to obtain t which maximizes p_*. Unfortunately, due to the complex structure of p_*, the maximum point p_* has no analytic expression. Therefore, to obtain the maximum point, one needs to apply an optimization algorithm such as gradient descent and Newton's method, which is inflexible. To make our selection easy to apply, we instead adopt the following heuristic approach to obtain t.

Consider a similar rejection sampling, which takes continuous Laplace $\mathcal{L}(t)$ as proposal distribution, and continuous Gaussian $\mathcal{N}(\sigma)$ as target distribution. According to a similar derivation of Section 4.4, we get the constant M' of this rejection sampling that

$$M' = \frac{c'_1}{c'_2} \cdot e^{\sigma^2/(2t^2)}$$

where c'_1 and c'_2 are the normalizing constants of $\mathcal{L}(t)$ and $\mathcal{N}(\sigma)$ respectively, such that

$$c'_1 = \frac{1}{\int_{-\infty}^{+\infty} e^{-|x|/t} \mathrm{d}x} = \frac{1}{2t} \quad c'_2 = \frac{1}{\int_{-\infty}^{+\infty} e^{-x^2/(2\sigma^2)} \mathrm{d}x} = \frac{1}{\sqrt{2\pi}\sigma}$$

Thus, the acceptance probability $p_\#$ of this rejection sampling is

$$p_\# = \frac{1}{M'} = \frac{\sqrt{2\pi}\sigma}{2t} \cdot e^{-\sigma^2/(2t^2)}$$

Our basic idea is that, since the discrete Laplace (Gaussian) distribution is the natural discrete analog of continuous Laplace (Gaussian) distribution, the accept probabilities p_* and $p_\#$ are similar to some extent. Therefore, we heuristically choose t as the maximum point of $p_\#$, which has an analytic expression. In particular, the partial derivative function of $p_\#$ with respect to t is:

$$\frac{\partial p_\#}{\partial t} = \frac{\sqrt{2\pi}\sigma}{2t^4} \cdot e^{-\sigma^2/(2t^2)} \cdot (\sigma^2 - t^2)$$

It is easy to see that $t = \sigma$ is the maximum point of $p_\#$. Therefore, we can choose $t = \sigma$ as our heuristic value.

Second, as described in Section 4.4, to apply the sampling circuit of $\mathcal{B}(\exp(-u/r))$ to sampling acceptance bit from $\mathcal{B}(f_*(s))$, we should select t to ensure $f_*(x)$ has the form that

$$f_*(x) = \exp\left(-\frac{(|x| - \sigma^2/t)^2}{2\sigma^2}\right) = \exp\left(-\frac{(|x| - z_0)^2}{2\sigma^2}\right) = \exp\left(-\frac{g_*(x)}{r}\right)$$

where $z_0 = \sigma^2/t$, $g_*(x) \geq 0$ is an integer function and $r \in \mathbb{R}^+$. A natural idea is to let $g_*(x) = (|x| - z_0)^2$ and $r = 2\sigma^2$. To ensure that $g_*(x)$ is an integer function, we should let $z_0 \in \mathbb{Z}^+$. But as previously discussed, we get $t = \sigma$ and $z_0 = \sigma^2/t = \sigma \in \mathbb{R}^+$. To ensure $z_0 \in \mathbb{Z}^+$, we fine-tune it to be $z_0 = \lfloor\sigma\rceil$, where $\lfloor\sigma\rceil$ means the nearest positive integer of σ. After fine-tuning, the value of t becomes $t = \sigma^2/z_0 = \sigma^2/\lfloor\sigma\rceil$. Proposition 4.2 shows that this selection achieves a high acceptance probability when $\sigma \geq 1$.

Proposition 4.2. *For $\sigma \geq 1$, $N \geq 2\lfloor\sigma\rceil$ and $t = \sigma^2/\lfloor\sigma\rceil$, we have the acceptance probability $p_* > 0.64$.*

The proof of Proposition 4.2 is shown in Appendix A.

However, when $\sigma \in (0, 1)$, the above selection of t leads to a poor acceptance probability, as shown in Table 2.

Table 2: The acceptance probability for $\sigma \in (0, 1)$ with $t = \sigma^2/\lfloor\sigma\rceil$

σ	0.1	0.2	0.3	0.4	0.5	0.6
p_*	1.9×10^{-22}	3.7×10^{-6}	0.0038	0.04	0.16	0.33

Intuitively, the reason is that the fine-tuning from $z_0 = \sigma \in (0, 1)$ to $z_0 = 1$ is somehow "far" and leads to a large performance degradation. To fix this problem, we give another selection of t for $\sigma \in (0, 1)$. Notice that $f_*(x)$ can be rewritten to:

$$f_*(x) = \exp\left(-\frac{(|x| - z_0)^2}{2\sigma^2}\right) = \exp\left(-\frac{(\frac{1}{z_0}|x| - 1)^2}{2\sigma^2 \frac{1}{z_0^2}}\right)$$

Thus we can let $g_*(x) = (\frac{1}{z_0}x - 1)^2$ and $r = 2\sigma^2 \frac{1}{z_0^2}$. To ensure that $g_*(x)$ is an integer function, we should let $\frac{1}{z_0} \in \mathbb{Z}^+$. Therefore, we can fine-tune $z_0 = \sigma \in (0, 1)$ to $z_0 = \frac{1}{z_1}$ for some $z_1 \in \mathbb{Z}^+$. Notice that this fine-tuning is "nearer" than set $z_0 = 1$. In particular, to let $z_1 = \frac{1}{z_0} \in \mathbb{Z}^+$, we can fine-tune it to be $z_1 = \lceil\frac{1}{\sigma}\rceil$ and get $z_0 = 1/\lceil\frac{1}{\sigma}\rceil$, where $\lceil\frac{1}{\sigma}\rceil$ is the smallest integer not less than $\frac{1}{\sigma}$. Therefore, the value of t becomes $t = \sigma^2/z_0 = \sigma^2\lceil\frac{1}{\sigma}\rceil$.

Figure 2 gives an example of these two fine-tuning processes with $\sigma = 0.4$. Before fine-tuning, the value of z_0 is 0.4. The previous fine-tuning process ($z_0 = \lfloor\sigma\rceil$) rounds z_0 to 1, which leads to a large degradation of acceptance probability (from 0.55 to 0.04). On the other hand, the new process ($z_0 = 1/\lceil\frac{1}{\sigma}\rceil$) fine-tunes z_0 to nearer $\frac{1}{3} \approx 0.33$, which keeps a high acceptance probability 0.59.

Proposition 4.3 shows that this new fine-tuning process achieves a high acceptance probability when $\sigma \in (0, 1)$.

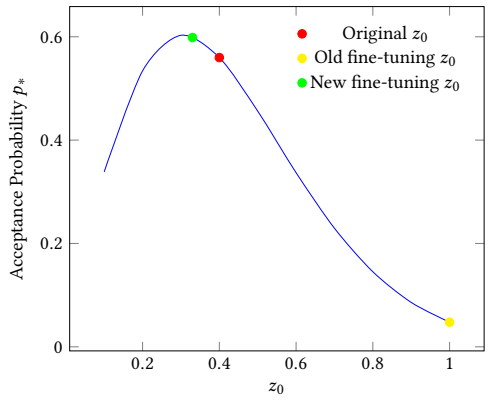

Figure 2: The acceptance probability for different chosen $z_0 = \sigma^2/t$ with $\sigma = 0.4$. The old fine-tuning process rounds original $z_0 = 0.4$ to $z_0 = 1$, which leads to a large degradation of acceptance probability (from 0.55 to 0.04). The new fine-tuning process selects a nearer $z_0 = \frac{1}{3} \approx 0.33$ and keeps a high acceptance probability 0.59.

Proposition 4.3. *For $\sigma \in (0, 1)$, $t = \sigma^2\lceil\frac{1}{\sigma}\rceil$, we have the acceptance probability $p_* > 0.54$.*

The proof of Proposition 4.3 is shown in Appendix A.

In summary, our selection of discrete Laplace parameter t is

$$t = \begin{cases} \sigma^2/\lfloor\sigma\rceil & if\ \sigma \geq 1 \\ \sigma^2\lceil 1/\sigma\rceil & if\ 0 < \sigma < 1 \end{cases}$$

And the corresponding $f_*(x) = \exp(-g_*(x)/r)$ becomes:

$$\begin{array}{lll} g_*(x) = (|x| - \lfloor\sigma\rceil)^2 & r = 2\sigma^2 & if\ \sigma \geq 1 \\ g_*(x) = (\lceil 1/\sigma\rceil|x| - 1)^2 & r = 2\sigma^2\lceil 1/\sigma\rceil^2 & if\ 0 < \sigma < 1 \end{array}$$

4.6 Sampling Discrete Gaussian Distribution

Combining the components in previous sections, we can construct our 2PC protocol for sampling discrete Gaussian distribution.

First, we construct the rejection sampling circuit for generating n independent discrete Gaussian samples with a fixed number of m trials. For each trial, we generate a discrete Laplace sample and the corresponding acceptance bit, based on Algorithms 3 and 4. The rejection sampling circuits are shown in Algorithm 5.

Then, we can apply a generic 2PC protocol to this rejection sampling circuit. Since our circuit mainly contains Boolean operations, we apply a garbled circuit-based generic 2PC protocol. After the 2PC, we obtain all m discrete Laplace samples and the corresponding acceptance bits in garbled form. Finally, we reveal all the acceptance bits in public and select the first n accepted samples based on the acceptance bits. The sampling process is shown in Algorithm 6.

Theorem 4.4. *Let σ be the discrete Gaussian parameter, and n be the number of samples. Denote $\mathcal{V}$ as the output distribution of Algorithm 6, and $\mathcal{N}_{\mathbb{Z}}^n(\sigma)$ as the n-dimensions distribution with each dimension independently sampled from $\mathcal{N}_{\mathbb{Z}}(\sigma)$. Then the statistical distance between $\mathcal{V}$ and $\mathcal{N}_{\mathbb{Z}}^n(\sigma)$ is bounded by*

$$\mathrm{SD}(\mathcal{V}, \mathcal{N}_{\mathbb{Z}}^n(\sigma)) \leq \delta_t + \delta_b + \delta_r$$

Algorithm 5: SamplingCircuit-$\mathcal{N}_{\mathbb{Z}}(\sigma, n)$

Input: Parameter σ, sample number n

Output: m samples $s_1, ..., s_m$ and the acceptance bits $b_1, ..., b_m$, where the accepted samples follow $\mathcal{N}_{\mathbb{Z}}(\sigma)$

Precompute: $\begin{cases} t = \sigma^2/\lfloor\sigma\rceil, r = 2\sigma^2 & if\ \sigma \geq 1 \\ t = \sigma^2\lceil 1/\sigma\rceil, r = 2\sigma^2\lceil 1/\sigma\rceil^2 & if\ 0 < \sigma < 1 \end{cases}$

for $i \leftarrow 1$ **to** m **do**
  $s_i \leftarrow$ **SamplingCircuit-**$\mathcal{L}_{\mathbb{Z},N}(t)$
  if $\sigma \geq 1$ **then**
    $v_i \leftarrow \mathbf{SUB}(\mathbf{ABS}(s_i), \lfloor\sigma\rceil)$
    $u_i \leftarrow \mathbf{SQUARE}(v_i)$ ▷ $u_i = (|s_i| - \lfloor\sigma\rceil)^2$
  else
    $v_i \leftarrow \mathbf{SUB}(\mathbf{MUL}(\lceil 1/\sigma\rceil, \mathbf{ABS}(s_i)), 1)$
    $u_i \leftarrow \mathbf{SQUARE}(v_i)$ ▷ $u_i = (\lceil 1/\sigma\rceil|s_i| - 1)^2$
  end
  $b_i \leftarrow$ **SamplingCircuit-**$\mathcal{B}_{\mathbf{exp}}(u_i, r)$
end
return $s = (s_1, s_2, ..., s_m), b = (b_1, b_2, ..., b_m)$

Algorithm 6: SecureSampling$\mathcal{N}_{\mathbb{Z}}(\sigma, n)$

Input: Parameter σ, sample number n

Output: n independent samples $x_1, ..., x_n$ from $\mathcal{N}_{\mathbb{Z}}(\sigma)$

$[[s]], [[b]] \leftarrow$ **MPC** SamplingCircuit-$\mathcal{N}_{\mathbb{Z}}(\sigma, n)$
$b \leftarrow \mathbf{Reveal}([[b]])$
Initialize x to an array of size n
for $i \leftarrow 1$ **to** m **do**
  if $b_i = 1$ **and** x *is not full* **then** add $[[s_i]]$ to x;
end
return $x = (x_1, x_2, ..., x_n)$

where

$$\delta_t = 2ne^{-\frac{N^2}{2\sigma^2}}$$
$$\delta_b = \frac{n}{p'_*} \cdot (2\kappa + l + 2) \cdot 2^{-\mu}$$
$$\delta_r = \Pr\left[X < n \mid X \sim \mathbf{Bin}(m, p'_*)\right]$$

with $p'_* = p_* - (2\kappa + l + 2) \cdot 2^{-\mu}$

Here $\delta_t, \delta_b, \delta_r$ represent the statistical distance caused by truncation, Bernoulli sampling, and rejection sampling respectively, and

- *$N = 2^\kappa + 1$ ($\kappa \in \mathbb{N}^+$) is the range parameter.*
- *μ is the number of precision bits in Bernoulli sampling.*
- *l is the number of bits to represent the output of $g_*(x)$, which is defined in Section 4.5.*
- *m is the number of trial times in rejection sampling.*
- *p_* is the acceptance probability of rejection sampling.*

and $\mathbf{Bin}(m, p)$ is the Binomial distribution with m trials and success probability p.

We defer the proof of Theorem 4.4 to Appendix B.

Selection of Parameters. To achieve total statistical distance $2^{-\lambda}$, we describe the selection of parameters in our algorithm. According to Theorem 4.4, since our algorithm introduces three addends

$\delta_t, \delta_b, \delta_r$ to statistical distance, it is a good idea to set $\delta_t \le 2^{-\lambda}/2$, $\delta_b \le 2^{-\lambda}/4$ and $\delta_r \le 2^{-\lambda}/4$ respectively. In particular, we choose the following parameters to satisfy the above requirement.

- N: According to $\delta_t \le 2ne^{-\frac{N^2}{2\sigma^2}} \le 2^{-\lambda}/2$, we get
$$N \ge \sqrt{2\ln 2(\lambda + 2 + \log n)} \cdot \sigma \triangleq N_0$$
Since N requires $N = 2^\kappa + 1$ ($\kappa \in \mathbb{N}^+$), we can choose
$$\kappa = \lceil \log(N_0 - 1) \rceil \text{ and } N = 2^{\lceil \log(N_0-1) \rceil} + 1$$
- l: According to Section 4.5, we have
$$g_*(x) = (|x| - \lfloor \sigma \rceil)^2 \qquad (\sigma \ge 1)$$
$$g_*(x) = (\lceil 1/\sigma \rceil |x| - 1)^2 \qquad (\sigma < 1)$$
Since x is ranged in $(-N, N)$ and $N \ge 2\lfloor \sigma \rceil$, we have $\max g_*(x) = g_*(N-1) = g_*(2^\kappa)$. Thus we can choose $l \ge \log g_*(2^\kappa)$, s.t.
$$l = 2\kappa \qquad (\sigma \ge 1)$$
$$l = \lceil 2(\kappa + 1) + 2\log(\lceil 1/\sigma \rceil) \rceil \qquad (\sigma < 1)$$
- μ: According to $\delta_b = \frac{n}{p'} \cdot (2\kappa + l + 2) \cdot 2^{-\mu} \le 2^{-\lambda}/4$, we get $(2\kappa + l + 2) \cdot 2^{-\mu} \le 2^{-\lambda} \cdot \frac{p'_*}{4n} \le 2^{-\lambda}$. Therefore, we have
$$p'_* = p_* - (2\kappa + l + 2) \cdot 2^{-\mu} \ge p_* - 2^{-\lambda} \triangleq p_0$$
We can choose μ as the the minimum integer that satisfies $(2\kappa + l + 2) \cdot 2^{-\mu} \le 2^{-\lambda} \cdot \frac{p_0}{4n}$, which leads to
$$\mu = \lceil \lambda + 2 + \log(n(2\kappa + l + 2)/p_0) \rceil$$
- m: Similar to μ, we choose m to satisfy:
$$\delta_r \le \Pr[X < n \mid X \sim \mathbf{Bin}(m, p_0)] \le 2^{-\lambda}/4$$
According to Hoeffding's inequality that
$$\Pr[X < n \mid X \sim \mathbf{Bin}(m, p_0)] \le \exp\left(-\frac{2(mp_0 - n)^2}{m}\right)$$
Thus we choose m as the minimum integer that satisfy: $\exp\left(-\frac{2(mp_0-n)^2}{m}\right) \le 2^{-\lambda}/4$. By solving it, we get
$$m = \left\lceil k_1 + k_2/2 + \sqrt{k_2^2/4 + k_1 k_2} \right\rceil$$
where $k_1 = n/p_0$ and $k_2 = (\lambda + 2)\ln 2/(2p_0^2)$.

4.7 Extension to Malicious Security and MPC

Extend to Malicious Security. To satisfy malicious security in which the parties may violate the protocol, we can apply a generic malicious (instead of semi-honest) 2PC protocol to our sampling circuit. In this case, the malicious input is prevented by the XOR technique, and other malicious behavior during computation is prevented by this malicious 2PC protocol. An example of the malicious 2PC protocol is the Yao protocol with *cut-and-choose* technique [9], in which the garbler P_1 sends many independent garbled circuits to evaluator P_2, P_2 then choose a random subset of these circuits, and ask P_1 to "open" them to verify the correctness. If all the opened circuits are correct, then P_2 evaluates the remaining circuit as in the standard Yao's protocol.

Extend to MPC. Though 2PC is sufficient in our system model, our solution can also be extended to MPC. When more than two parties $P_1, P_2, ..., P_n$ join in the protocol, we can also ask each party P_i to provide uniform random bits r_i as input. Similarly, by XORing the input from each party, one can get the correct uniform bits as long as one of the parties provides the actual randomness. Then, we can apply a generic MPC protocol (e.g. BMR [3]) to compute the function $\mathbf{Noise}(r_1, r_2, ..., r_n) = f(r_1 \oplus r_2 \oplus ... \oplus r_n)$, where f is our designed sampling method.

5 PRIVACY

Consider a DP mechanism $L : \mathbb{F}^d \to \mathbb{Z}^n$ based on true discrete Gaussian distribution, such that: $L(x) = q(x) + Y$, where Y_i is independently drawn from $\mathcal{N}_\mathbb{Z}(\sigma)$. According to [7], L satisfies (ϵ, δ)-DP iff for all neighbouring $D, D' \in \mathbb{F}^d$:
$$\delta \ge \Pr[Z > \epsilon] - e^\epsilon \cdot \Pr[Z < -\epsilon]$$
where
$$Z = \frac{\sum_{j=1}^n (q(D)_j - q(D')_j)^2 + 2(q(D)_j - q(D')_j) \cdot Y_j}{2\sigma^2}$$

Next, we consider the mechanism $\widetilde{L}$, which remains the same as L except the noise Y is generated by Algorithm 6. We show the differential privacy of $\widetilde{L}$.

Theorem 5.1. *For $\epsilon, \delta > 0$, if L satisfies (ϵ, δ)-DP, then $\widetilde{L}$ satisfies $(\epsilon, \delta + \delta')$-DP, where*
$$\delta' = 2(e^\epsilon + 1)(\delta_t + \delta_b + \delta_r)$$
with δ_t, δ_b, δ_r defined in Theorem 4.4.

Proof: For any $\mathbb{S} \subset \mathbb{Z}^n$ and any neighbouring $D, D' \in \mathbb{F}^n$, the differential privacy of L follows that
$$\Pr[L(D) \in \mathbb{S}] - e^\epsilon \Pr\left[L(D') \in \mathbb{S}\right] \le \delta \tag{1}$$
and according to Theorem 4.4, we have
$$\begin{aligned} \left|\Pr\left[\widetilde{L}(D) \in \mathbb{S}\right] - \Pr[L(D) \in \mathbb{S}]\right| &\le 2(\delta_t + \delta_b + \delta_r) \\ \left|\Pr\left[\widetilde{L}(D') \in \mathbb{S}\right] - \Pr[L(D') \in \mathbb{S}]\right| &\le 2(\delta_t + \delta_b + \delta_r) \end{aligned} \tag{2}$$
Based on Equation (1), Equation (2), we have
$$\begin{aligned} &\Pr\left[\widetilde{L}(D) \in \mathbb{S}\right] - e^\epsilon \cdot \Pr\left[\widetilde{L}(D') \in \mathbb{S}\right] \\ &\le \Pr[L(D) \in \mathbb{S}] - e^\epsilon \Pr\left[L(D') \in \mathbb{S}\right] \\ &\quad + \left|\Pr\left[\widetilde{L}(D) \in \mathbb{S}\right] - \Pr[L(D) \in \mathbb{S}]\right| \\ &\quad + e^\epsilon \left|\Pr\left[\widetilde{L}(D') \in \mathbb{S}\right] - \Pr\left[L(D') \in \mathbb{S}\right]\right| \\ &\le \delta + 2(e^\epsilon + 1)(\delta_t + \delta_b + \delta_r) = \delta + \delta' \end{aligned}$$

6 COMPLEXITY

We present the complexity analysis on our discrete Gaussian sampling circuit, i.e. Algorithms 5. In the following analysis, the circuit complexity and the cost refer to the number of AND gates.

Theorem 6.1. *Let σ be the discrete Gaussian parameters, n be the number of samples, λ be the security parameters. Denote $\psi = \lambda + \log n$. The average circuit complexity of* **SamplingCircuit**-$\mathcal{N}_\mathbb{Z}(\sigma, n)$ *for generating each sample is $O(\psi \log(\psi\sigma))$.*

Proof: According to Section 4.6, to achieve total statistical distance $2^{-\lambda}$, the parameters in our algorithm becomes

$$\kappa = O(\log(\psi\sigma)) \quad l = O(\log(\psi\sigma))$$
$$\mu = O(\psi) \quad m = O(n+\lambda)$$

The cost of our sampling circuit mainly includes two parts (1) the cost C_1 to generate all Bernoulli samples and (2) the cost C_2 for m evaluations of $g_*(x)$ (defined in Section 4.5).

For C_1, recall that our circuits generate $(\kappa + l + 1)$ Bernoulli samples in each trial, ($\kappa + 1$ for sampling discrete Laplace, l for sampling acceptance bits), each with μ bits of precision, Thus, we have

$$C_1 = m \cdot (\kappa + l + 1) \cdot \mu = O((n+\lambda)\log(\psi\sigma)\psi)$$

For C_2, notice that the main cost of the computation of $g_*(x)$ is the $O(\kappa)$-bits multiplication, which leads to

$$C_2 = O(m\kappa^2) = O((n+\lambda)\log^2(\psi\sigma))$$

By combining C_1 and C_2, the average cost $\overline{C}$ of the whole sampling circuit for each sample is

$$\overline{C} = (C_1 + C_2)/n = O(\log(\psi\sigma)\psi)$$

7 EXPERIMENTS

7.1 Environment

We implement our discrete Gaussian sampling method based on Yao's protocol in MP-SPDZ framework [29]. The evaluation is performed in two identical computers each with 2× Intel Xeon Platinum 8369B 2.7GHz, Ubuntu 20.04 , and 4GB memory. The bandwidth between these two instances is 10Gbits per second.

7.2 Evaluation on Acceptance Probability

In this section, we evaluate the acceptance probability of our rejection sampling method (i.e. with discrete Laplace distribution as proposal distribution) for different discrete Gaussian parameters σ. For comparison, we also evaluate the acceptance probability with discrete uniform distribution as proposal distribution. Figure 3 shows the results. The acceptance probability of discrete uniform distribution is very small (< 0.15). On the other hand, the acceptance probability of our method is approximately 0.6 when $\sigma < 1$ and larger than 0.6 when $\sigma \geq 1$. In particular, when $\sigma \geq 5$, the acceptance probability of our method is stable at approximately 0.76. It shows that our chosen discrete Laplace has a great advantage in acceptance probability for sampling discrete Gaussian distribution.

7.3 Evaluation on Discrete Laplace Parameter t

In this section, we evaluate the AND gates of our sampling method with different discrete Laplace parameters t. As we describe in Section 4.5, in order to apply Algorithm 4 for sampling the accept bit, we evaluate t that satisfies $z_0 \in \mathbb{Z}^+$ or $\frac{1}{z_0} \in \mathbb{Z}^+$, where $z_0 = \sigma^2/t$. We evaluate four cases of $\sigma = 0.5, 1, 5, 10$ with the number of samples $n = 4096$ and security parameter $\lambda = 64$. Figure 4 shows the results. In all four cases, our selected t achieves minimum AND gates compared to other satisfied t. In particular, for giving $\sigma = 5$, the number of AND gates of our selected $t = 5$ is 3× smaller than $t \approx 2.2$ and 1.6× smaller than $t = 12.5$.

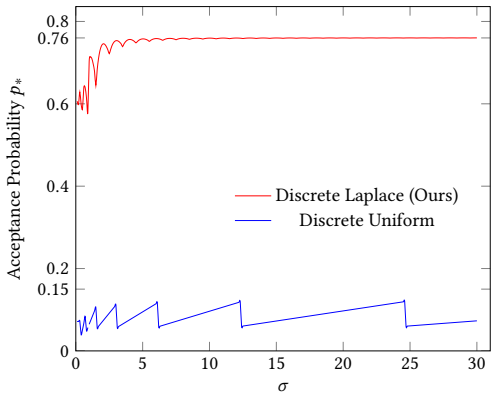

Figure 3: The acceptance probability of discrete Laplace and discrete uniform with different discrete Gaussian parameters σ. The acceptance probability of discrete uniform is very small (< 0.15). And the discrete Laplace with our selected parameter t achieves a high acceptance probability.

Table 3: Number of AND gates of (1) our method for generating $n = 4096$ discrete Gaussian samples (2) the Box-Muller method for generating $n = 4096$ continuous Gaussian samples (3) the computation on the exponential function (EXP) $\exp(x)$ with a random x for $n = 4096$ times. Here we fix the security parameter $\lambda = 64$ and Gaussian parameter $\sigma = 5$.

Method	Ours	Box-Muller	EXP
AND gates ($\times 10^6$)	10.0	4951.4	6501.9

7.4 Evaluation on Discrete Gaussian Sampling

In this section, we evaluate our discrete Gaussian sampling method and compare it with other methods. The security model is the same as our methods defined in Section 3.1 such that the parties will infer the private information and poison the result.

First, to illustrate the inefficiency of transcendent functions (i.e. exponential, trigonometric), we compare the number of AND gates of our method for generating $n = 4096$ discrete Gaussian samples with (1) the Box-Muller method (which includes trigonometric and logarithmic computation) for generating $n = 4096$ continuous Gaussian samples and (2) the computation on the exponential function (EXP) $\exp(x)$ with a random x for $n = 4096$ times. Here we fix the Gaussian parameter $\sigma = 5$ and security parameter $\lambda = 64$. The result is shown in Table 3. The overhead of the Box-Muller method and EXP are 500× and 650× respectively compared to our method.

Then, we compare our method with other sampling methods for discrete Gaussian. We implement some other common discrete Gaussian sampling methods (in ordinary sampling) in MPC, including (1) the cumulative distributed table-based sampling (CDT) (2) the Kunth-Yao sampling (KY) (3) the rejection sampling with discrete uniform as proposal distribution (UNI). These methods cannot be directly used for our security model, therefore, we implement them by (1) applying the XOR technique to get the correct uniform bits (2) converting the origin sampling method to a circuit and applying the Yao protocol. Besides, we also implement and evaluate the distributed noise generation (DNG), with an additional statistical test on inputs. Table 4 shows the number of AND gates of different sampling methods for generating $n = 4096$ discrete Gaussian samples with $\sigma \in [0.1, 40]$ and $\lambda = 128$. The CDT and DNG

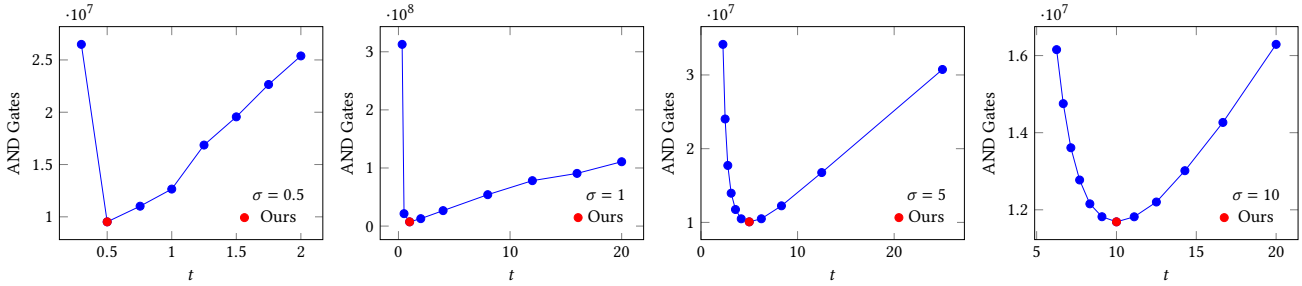

Figure 4: Number of AND gates for our sampling method with different chosen discrete Laplace parameters t for given discrete Gaussian parameter $\sigma = 0.5, 1, 5, 10$. Here we fix number of sample $n = 2^{12} = 4096$ and security parameter $\lambda = 64$. In all cases, our selected t achieves minimum gate complexity.

Table 4: Number of AND Gates of different methods including (1) Our method (Ours) (2) the cumulative distributed table-based sampling (CDT) (3) the Kunth-Yao sampling (KY) (4) the rejection sampling with discrete uniform as proposal distribution (UNI) (5) distributed noise generation (DNG) with an statistical test for malicious security. Results are measured when generating $n = 4096$ discrete Gaussian samples with security parameter $\lambda = 128$.

AND gates ($\times 10^6$) \ σ / Method	$\sigma = 0.1$	$\sigma = 0.5$	$\sigma = 1$	$\sigma = 5$	$\sigma = 10$	$\sigma = 20$	$\sigma = 40$
Ours	16.6	17.0	13.0	**20.7**	**23.5**	**36.4**	**29.3**
CDT	1.8	4.9	9.4	47.4	95.8	194.5	395.9
UNI	50.8	114.8	116.4	307.7	345.4	386.0	428.5
KY	44.2	139.5	294.2	1828.4	3994.2	8699.7	18905.0
DNG	**1.4**	**4.5**	**9.1**	51.5	107.3	223.4	465.6

methods have a great performance for small $\sigma \leq 1$. The reason is that the most consumed part of these two methods is the $O(N)$ linear scan over all the elements $(-N, N) \cap \mathbb{Z}$ in the distribution. And when σ is small, the required N is also small. On the other hand, the number of AND gates of our method have a slow growth rate with respect to σ, and for slightly large $\sigma \geq 5$, our method has the least AND gates among all methods. In particular, for $\sigma = 10$, the number of AND gates of our method are 4×, 5×, 15×, 170× smaller compared to CDT, DNG UNI, and KY methods respectively.

According to Table 4, the CDT method has the best performance among other methods when σ is large. Thus, we perform a further comparison with the CDT method. Figure 5 shows the results for generating $n = 2^{14}$ samples with different σ and $\lambda \in \{64, 96, 128\}$. The cost of our method grows at a slower rate with σ and has a better performance when $\sigma \geq 5$. It shows that our method is efficient even for large σ. Figure 6 shows the results for different sampling numbers $n \in [2^{12}, 2^{15}]$ with $\sigma = 20$ and $\lambda \in \{64, 96, 128\}$. In this case, our algorithm achieves 4.5× speedups in execution time and 8× reduction in communication size. In particular, our method generates 2^{15} samples with $\sigma = 20$ and $\lambda = 128$ in 1.5 minutes, while the CDT method costs 8 minutes for the same generation.

8 RELATED WORK

Several works [6, 21, 22, 24] suggested using distributed noise generation for DP in MPC, where each party locally computes partial noises and securely combines them. Distributed noise generation is efficient since only the light addition operations are executed in MPC. However, distributed noise generation in the presence of malicious parties is not possible without additional checks [6].

Some other works use the sampling method in MPC that can be naturally represented as a circuit. Therefore, by incorporating the XOR technique, one can easily construct a malicious secure solution. For example, Eigner et al. [16] sample Laplace based on inverse transform sampling, and Pentyala et al. [33] sample continuous Gaussian based on the Box-Muller method. However, these methods include floating-point arithmetic and transcendental functions which are extremely complicated in MPC. For sampling discrete Laplace, Dwork et al. [15] observed that the sampling of geometric distribution (the half distribution) can be reduced to sampling Bernoulli distribution for each bit, which only includes efficient bit operations. Champion et al. [8] applied an oblivious stack to improve the Bernoulli sampling circuits, which can be used to construct a more efficient sampling based on Dwork's solution. Influencing by these works, our work utilizes the efficient sampling of discrete Laplace to sample discrete Gaussian.

Discrete Gaussian sampling is usually used for the lattice-based cryptography system. As discussed in Section 4, if a circuit of a sampling method is designed, we can easily implement a malicious solution by incorporating the XOR technique. The common types of sampling methods include table-based sampling and rejection sampling. The table-based sampling performs an efficient search e.g., binary search on precomputed tables, which includes CDT-based sampling [32], Knuth-Yao sampling [14], binary arithmetic coding (BAC) sampling [36], etc. To convert the table-based sampling to a circuit, one should instead perform an expensive linear scan on the

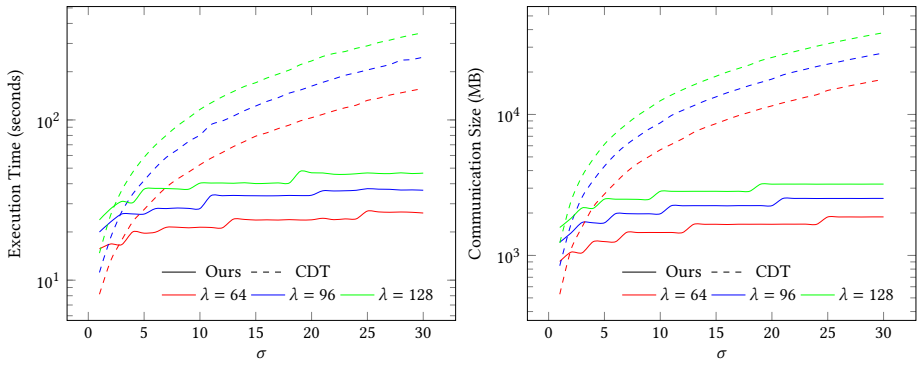

Figure 5: The execution time and communication size of our method and CDT method with $n = 16384$, $\lambda = \{64, 96, 128\}$.

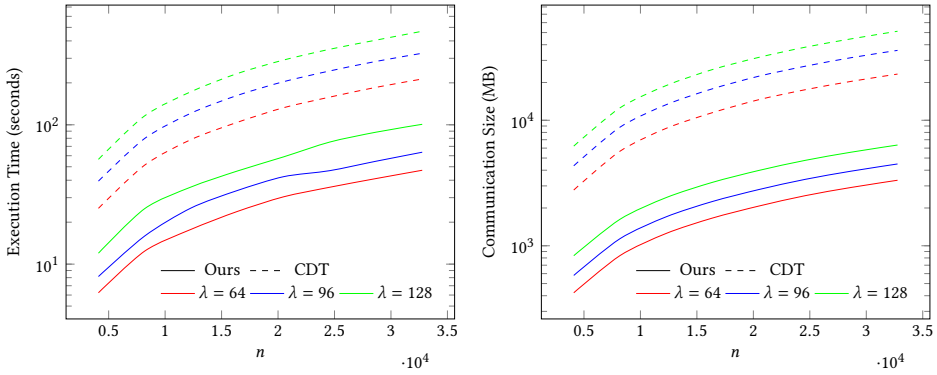

Figure 6: The execution time and communication size of our method and CDT method with $\sigma = 20$, $\lambda = \{64, 96, 128\}$.

table in order not to leak privacy. Compared to table-based sampling, rejection sampling can utilize the easier proposal distribution to sample the target distribution. However, the sampling of the acceptance bit usually involves expensive exponential computation (e.g., [19]). To solve this problem, several works apply techniques including lazy sampling [13], von Neumann's algorithm [27], and Bernoulli combination [12]. The idea of lazy sampling is to perform a cheap computation for some simple cases, and only perform the exponential computation in the rest cases. However, this process can not benefit the circuit since each computation flow must be executed in order not to leak privacy. Von Neumann's algorithm uses a complicated computation flow which is not suitable to convert into a circuit. The Bernoulli combination technique decomposes the sampling of acceptance bit into sampling several Bernoulli variables, which can be naturally converted into a circuit and therefore our work follows this technique. Moreover, Karmakar et al. [25, 26] designed a discrete Gaussian sampling method based on computing a series of Boolean functions, which is very efficient but requires a huge codebase to describe all the Boolean functions.

9 CONCLUSION

In summary, this work presents a new solution for implementing Differential Privacy (DP) using Secure Multi-Party Computation (MPC) without requiring a trusted third party. The focus is on discrete Gaussian noise sampling, which is an important aspect of implementing DP mechanisms in MPC. There is currently no solution for discrete Gaussian sampling that plays nicely with malicious secure MPC protocol and our work fills this gap. We use a rejection sampling approach, sampling a proposal distribution and determining whether to accept the sample. To ensure security against data poisoning attacks, we require each party to provide uniform random bits as input. We implement and evaluate our sampling method in MP-SPDZ framework [29]. The results show that our method can generate $2^{15} = 32768$ discrete Gaussian samples with $\sigma = 20$ and security parameter $\lambda = 128$ in 1.5 minutes.

ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their insightful comments. This work is supported in part by the National Natural Science Foundation of China (NSFC) under No.62302441, the Funding for Postdoctoral Scientific Research Projects in Zhejiang Province (ZJ2022072), ZJU–DAS-Security Joint Research Institute of Frontier Technologies and the Supercomputing Center of Hangzhou City University. Tianhao did part of the work while part-time at Meta.

REFERENCES

[1] Balamurugan Anandan and Chris Clifton. 2015. Laplace noise generation for two-party computational differential privacy. In *2015 13th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 54–61.

[2] Victor Balcer and Albert Cheu. 2019. Separating local & shuffled differential privacy via histograms. *arXiv preprint arXiv:1911.06879* (2019).

[3] D Beaver, S Micali, and P Rogaway. The round complexity of secure protocols extended abstract. In *22nd ACM STOC*.

[4] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th symposium on operating systems principles*. 441–459.

[5] Jonas Böhler and Florian Kerschbaum. 2020. Secure multi-party computation of differentially private median. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2147–2164.

[6] Jonas Böhler and Florian Kerschbaum. 2021. Secure Multi-party Computation of Differentially Private Heavy Hitters. (2021).

[7] Clément L Canonne, Gautam Kamath, and Thomas Steinke. 2020. The discrete gaussian for differential privacy. *arXiv preprint arXiv:2004.00010* (2020).

[8] Jeffrey Champion, Abhi Shelat, and Jonathan Ullman. 2019. Securely sampling biased coins with applications to differential privacy. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 603–614.

[9] David Chaum. 1984. Blind signature system. In *Advances in Cryptology: Proceedings of Crypto 83*. Springer, 153–153.

[10] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. 2019. Distributed differential privacy via shuffling. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 375–403.

[11] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting telemetry data privately. *Advances in Neural Information Processing Systems* 30 (2017).

[12] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 2013. Lattice signatures and bimodal Gaussians. In *Annual Cryptology Conference*. Springer, 40–56.

[13] Léo Ducas and Phong Q Nguyen. 2012. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 415–432.

[14] Nagarjun C Dwarakanath and Steven D Galbraith. 2014. Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing* 25, 3 (2014), 159–180.

[15] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 486–503.

[16] Fabienne Eigner, Aniket Kate, Matteo Maffei, Francesca Pampaloni, and Ivan Pryvalov. 2014. Differentially private data aggregation with optimal utility. In *Proceedings of the 30th Annual Computer Security Applications Conference*. 316–325.

[17] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 1054–1067.

[18] David Evans, Vladimir Kolesnikov, Mike Rosulek, et al. 2018. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security* 2, 2-3 (2018), 70–246.

[19] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. 2008. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 197–206.

[20] Oded Goldreich, Silvio Micali, and Avi Wigderson. 2019. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 307–328.

[21] Slawomir Goryczka and Li Xiong. 2015. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE transactions on dependable and secure computing* 14, 5 (2015), 463–477.

[22] Mikko Heikkilä, Eemil Lagerspetz, Samuel Kaski, Kana Shimizu, Sasu Tarkoma, and Antti Honkela. 2017. Differentially private bayesian learning on distributed data. *Advances in neural information processing systems* 30 (2017).

[23] Jiankai Jin, Eleanor McMurtry, Benjamin IP Rubinstein, and Olga Ohrimenko. 2022. Are we there yet? timing and floating-point attacks on differential privacy systems. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 473–488.

[24] Peter Kairouz, Ziyu Liu, and Thomas Steinke. 2021. The distributed discrete gaussian mechanism for federated learning with secure aggregation. *arXiv preprint arXiv:2102.06387* (2021).

[25] Angshuman Karmakar, Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. 2018. Constant-time discrete gaussian sampling. *IEEE Trans. Comput.* 67, 11 (2018), 1561–1571.

[26] Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. 2019. Pushing the speed limit of constant-time discrete Gaussian sampling. A case study on the Falcon signature scheme. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[27] Charles FF Karney. 2016. Sampling exactly from the normal distribution. *ACM Transactions on Mathematical Software (TOMS)* 42, 1 (2016), 1–14.

[28] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.

[29] Marcel Keller. 2020. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 1575–1590.

[30] Andrew McGregor, Ilya Mironov, Toniann Pitassi, Omer Reingold, Kunal Talwar, and Salil Vadhan. 2010. The limits of two-party differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, 81–90.

[31] Ilya Mironov. 2012. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 650–661.

[32] Chris Peikert. 2010. An efficient and parallel Gaussian sampler for lattices. In *Annual Cryptology Conference*. Springer, 80–97.

[33] Sikha Pentyala, Davis Railsback, Ricardo Maia, Rafael Dowsley, David Melanson, Anderson Nascimento, and Martine De Cock. 2022. Training Differentially Private Models with Secure Multiparty Computation. *arXiv preprint arXiv:2202.02625* (2022).

[34] Ryan Rogers. 2020. A Differentially Private Data Analytics {API} at Scale. In *2020 {USENIX} Conference on Privacy Engineering Practice and Respect ({PEPR} 20)*.

[35] R Rogers, S Subramaniam, S Peng, D Durfee, S Lee, Santosh Kumar Kancha, S Sahay, P Ahammad, and API LinkedIn's Audience Engagements. 2020. A privacy preserving data analytics system at scale. *Linkedin's audience engagements api* (2020).

[36] Markku-Juhani O Saarinen. 2018. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering* 8, 1 (2018), 71–84.

[37] Apple's Differential Privacy Team. 2017. Learning with privacy at scale. (2017).

[38] WWDC. 2016. Engineering privacy for your users. (2016).

[39] Andrew C Yao. 1982. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 160–164.

A PROOF OF PROPOSITION 4.2 AND 4.3

We first prove that for $N \geq 2\sigma^2/t$:

$$\frac{\sum_{|n|<N} e^{-n^2/(2\sigma^2)}}{\sum_{|n|<N} e^{-|n|/t}} \geq \frac{\sum_{n\in\mathbb{Z}} e^{-n^2/2\sigma^2}}{\sum_{n\in\mathbb{Z}} e^{-|n|/t}}$$

Denote $a(n) \triangleq e^{-|n|/t}$ and $b(n) \triangleq e^{-n^2/(2\sigma^2)}$, it is equivalent to proving that

$$\frac{\sum_{n\in\mathbb{Z}} b(n)}{\sum_{n\in\mathbb{Z}} a(n)} \leq \frac{\sum_{|n|<N} b(n)}{\sum_{|n|<N} a(n)}$$

Consider the ratio between $b(n)$ and $a(n)$:

$$c(n) = \frac{b(n)}{a(n)} = \exp\left(\frac{-(|n| - \sigma^2/t)^2}{2\sigma^2} + \frac{\sigma^2}{2t^2}\right)$$

then we have for $m \geq 2\sigma^2/t$, $\max\limits_{|n|\geq m} c(n) = \min\limits_{|n|\leq m} c(n) = c(m)$. Therefore, for $N \geq 2\sigma^2/t$, we have

$$\frac{\sum_{|n|<N} b(n)}{\sum_{|n|<N} a(n)} \geq \min_{|n|<N} c(n) \geq \min_{|n|\leq N} c(n) = c(N)$$

$$\frac{\sum_{|n|\geq N} b(n)}{\sum_{|n|\geq N} a(n)} \leq \max_{|n|\geq N} c(n) = c(N)$$

which follows $\frac{\sum_{|n|<N} b(n)}{\sum_{|n|<N} a(n)} \geq \frac{\sum_{|n|\geq N} b(n)}{\sum_{|n|\geq N} a(n)}$, thus

$$\frac{\sum_{n\in\mathbb{Z}} b(n)}{\sum_{n\in\mathbb{Z}} a(n)} = \frac{\sum_{|n|<N} b(n) + \sum_{|n|\geq N} b(n)}{\sum_{|n|<N} a(n) + \sum_{|n|\geq N} a(n)} \leq \frac{\sum_{|n|<N} b(n)}{\sum_{|n|<N} a(n)}$$

Now we consider to estimate p_*, denote:

$$I(t) \triangleq \sum_{n\in\mathbb{Z}} e^{-|n|/t}, J(\sigma) \triangleq \sum_{n\in\mathbb{Z}} e^{-n^2/(2\sigma^2)}$$

$$S(t,\sigma) \triangleq e^{-\sigma^2/(2t^2)}$$

then we have, for $N \geq 2\sigma^2/t$

$$\begin{aligned} p_* &= \frac{\sum_{|n|<N} e^{-n^2/(2\sigma^2)}}{\sum_{|n|<N} e^{-|n|/t}} \cdot e^{-\sigma^2/(2t^2)} \\ &\geq \frac{\sum_{n\in\mathbb{Z}} e^{-n^2/(2\sigma^2)}}{\sum_{n\in\mathbb{Z}} e^{-|n|/t}} \cdot e^{-\sigma^2/(2t^2)} = \frac{J(\sigma)}{I(t)} \cdot S(t,\sigma) \end{aligned}$$

based on the inequalities $J(\sigma) \geq \max(1, \sqrt{2\pi}\sigma)$, we get:

$$p_* \geq \frac{S(t,\sigma)}{I(t)} \cdot \max(1, \sqrt{2\pi}\sigma) \tag{3}$$

For $\sigma \geq 1, t = \sigma^2/\lfloor\sigma\rceil$ and $N \geq 2\lfloor\sigma\rceil$, in this case $N \geq 2\sigma^2/t$ is satisfied, based on Equation (3) we have:

$$p_* \geq \frac{S(\sigma^2/\lfloor\sigma\rceil,\sigma)}{I(\sigma^2/\lfloor\sigma\rceil)} \cdot \sqrt{2\pi}\sigma \triangleq g_0(\sigma)$$

since $g_0(\sigma)$ is convex in $(n-1/2, n+1/2]$ for $n \in \mathbb{N}^+$, the possible minimum points are 1, 1.5^-, 1.5^+, 2.5^-, 2.5^+, 3.5^-, 3.5^+... . We construct two auxiliary functions:

$$g_1(\sigma) \triangleq= \frac{S(\sigma^2/(\sigma-1/2),\sigma)}{I(\sigma^2/(\sigma-1/2))} \cdot \sqrt{2\pi}\sigma$$

$$g_2(\sigma) \triangleq= \frac{S(\sigma^2/(\sigma+1/2),\sigma)}{I(\sigma^2/(\sigma+1/2))} \cdot \sqrt{2\pi}\sigma$$

which satisfy

$$g_0((n+1/2)^-) = g_1(n+1/2) \quad g_0((n+1/2)^+) = g_2(n+1/2)$$

for $n \in \mathbb{N}^+$. Notice that both $g_1(\sigma)$ and $g_2(\sigma)$ are monotonically increasing when $\sigma \geq 1$, which means that

$$g_0(1.5^-) < g_0(2.5^-) < g_0(3.5^-) < ...$$
$$g_0(1.5^+) < g_0(2.5^+) < g_0(3.5^+) < ...$$

Thus the possible minimum points are $1, 1.5^-$ and 1.5^+. We calculate and compare these three values and get the lower bound of p_*

$$p_* \geq \min_{\sigma\geq 1} g_0(\sigma) = \min(g_0(1), g_0(1.5^-), g_0(1.5^+)) > 0.64$$

Therefore we have $p_* > 0.64$ when $\sigma \geq 1$ and $N \geq 2\lfloor\sigma\rceil$.

For $\sigma \in (0,1)$ and $t = \sigma^2\lceil 1/\sigma\rceil$, in this case $N \geq 2\sigma^2/t$ is always satisfied. We consider $\sigma \in (0,1)$ in three intervals $[\frac{1}{\sqrt{2\pi}}, 1)$, $[\frac{1}{4}, \frac{1}{\sqrt{2\pi}}]$ and $(0, \frac{1}{4})$ respectively.

For $\sigma \in [\frac{1}{\sqrt{2\pi}}, 1)$, based on Equation (3) we have:

$$p_* \geq \frac{S(\sigma^2\lceil 1/\sigma\rceil,\sigma)}{I(\sigma^2\lceil 1/\sigma\rceil)} \cdot \sqrt{2\pi}\sigma \triangleq h_0(\sigma)$$

since $h_0(\sigma)$ is convex in $[\frac{1}{\sqrt{2\pi}}, \frac{1}{2})$ and $[\frac{1}{2}, 1)$, the possible minimum points are $\frac{1}{\sqrt{2\pi}}^+$, $\frac{1}{2}^-$, $\frac{1}{2}^+$ and 1^-. We calculate and compare these four values and get:

$$\begin{aligned} p_* &\geq \min_{\sigma\in[\frac{1}{\sqrt{2\pi}},1)} h_0(\sigma) \\ &= \min(h_0(\frac{1}{\sqrt{2\pi}}^+), h_0(\frac{1}{2}^-), h_0(\frac{1}{2}^+), h_0(1^-)) > 0.54 \end{aligned} \tag{4}$$

For $\sigma \in [\frac{1}{4}, \frac{1}{\sqrt{2\pi}})$, based on Equation (3) we have:

$$p_* \geq \frac{S(\sigma^2\lceil 1/\sigma\rceil,\sigma)}{I(\sigma^2\lceil 1/\sigma\rceil)} \triangleq h_2(\sigma)$$

Since $h_1(\sigma)$ is convex in $[\frac{1}{3}, \frac{1}{\sqrt{2\pi}})$ and $[\frac{1}{4}, \frac{1}{3})$, the possible minimum points are $\frac{1}{\sqrt{2\pi}}^-$, $\frac{1}{3}^-$, $\frac{1}{3}^+$ and $\frac{1}{4}^+$, we calculate and compare this four values and get:

$$\begin{aligned} p_* &\geq \min_{\sigma\in[\frac{1}{4},\frac{1}{\sqrt{2\pi}})} h_1(\sigma) \\ &= \min(h_1(\frac{1}{\sqrt{2\pi}}^-), h_1(\frac{1}{3}^-), h_1(\frac{1}{3}^+), h_1(\frac{1}{4}^+)) > 0.54 \end{aligned} \tag{5}$$

For $\sigma \in (0, \frac{1}{4})$, based on Equation (3) we have:

$$p_* \geq \frac{S(\sigma^2\lceil 1/\sigma\rceil,\sigma)}{I(\sigma^2\lceil 1/\sigma\rceil)}$$

notice that $S(t,\sigma)$ is monotonically increasing with t, and $I(t)$ is monotonically decreasing with t, then:

$$p_* \geq \frac{S(\sigma^2\lceil 1/\sigma\rceil,\sigma)}{I(\sigma^2\lceil 1/\sigma\rceil)} \geq \frac{S(\sigma^2(1/\sigma),\sigma)}{I(\sigma^2(1/\sigma+1))} = \frac{S(\sigma,\sigma)}{I(\sigma^2+\sigma)} \triangleq h_2(\sigma)$$

since $h_2(\sigma)$ is monotonically decreasing when $0 < \sigma < 1$, we have:

$$p_* \geq \min_{\sigma\in(0,\frac{1}{4})} h_2(\sigma) = h_2(\frac{1}{4}^-) > 0.55 \tag{6}$$

According to Equation (4), Equation (5), Equation (6), we have $p_* > 0.54$ when $\sigma \in (0,1)$.

B PROOF OF THEOREM 4.4

According to Section 2.4, we obtain that

$$\mathrm{SD}(\mathcal{N}^n_{\mathbb{Z}}(\sigma), \mathcal{N}^n_{\mathbb{Z},N}(\sigma)) \leq 2ne^{-\frac{N^2}{2\sigma^2}} \tag{7}$$

First, we consider the statistical distance between $\mathcal{G}_{2^\kappa}$ and the output distribution $\mathcal{G}'$ of Algorithm 2. Notice that the sample of $\mathcal{G}_{2^\kappa}$ can be seen as κ independent Bernoulli samples, with each sample having at most $2^{-\mu}$ statistical distance based on Algorithm 2. Therefore, we have:

$$\mathrm{SD}(\mathcal{G}'_{2^\kappa}, \mathcal{G}_{2^\kappa}) \leq \kappa \cdot 2^{-\mu} \tag{8}$$

Denote $N = 2^\kappa + 1$, we consider statistical distance between $\mathcal{L}_{\mathbb{Z},N}$ and the output distribution $\mathcal{L}'$ of Algorithm 3. Notice that we use a Bernoulli sample to determine whether to output 0 or other values, then we have:

$$f_{\mathcal{L}'}(0) = f_{\mathcal{L}_{\mathbb{Z},N}}(0) + \delta_d$$

for some $|\delta_d| \leq 2^{-\mu}$ and $f_{\mathcal{L}_{\mathbb{Z},N}}(0) + \delta_d \in [0,1]$. Then for $x \neq 0$, we have

$$f_{\mathcal{L}'}(x) = \frac{1}{2}(1 - f_{\mathcal{L}_{\mathbb{Z},N}}(0) - \delta_d) \cdot f_{\mathcal{G}_{2^\kappa}}(|x| - 1)$$

According to Equation (8), we have:

$$f_{\mathcal{G}'}(x) = f_{\mathcal{G}_{2^\kappa}}(x) + \delta_g(x)$$

with $\sum_x |\delta_g(x)| \le 2\kappa \cdot 2^{-\mu}$. Then

$$\begin{aligned}
\mathrm{SD}(\mathcal{L}_{\mathbb{Z},N}, \mathcal{L}') &= \frac{1}{2}\sum_x |f_{\mathcal{L}'}(x) - f_{\mathcal{L}}(x)| \\
&= \frac{1}{2}(|f_{\mathcal{L}'}(0) - f_{\mathcal{L}_{\mathbb{Z},N}}(0)| + \sum_{x\neq 0} |f_{\mathcal{L}'}(x) - f_{\mathcal{L}_{\mathbb{Z},N}}(x)|) \\
&\le \frac{1}{2}\left(|\delta_d| + (1 - f_{\mathcal{L}_{\mathbb{Z},N}}(0) - \delta_b)\sum_x |\delta_g(x)| + |\delta_d| \sum_x f_{\mathcal{G}_{2^\kappa}}(x)\right) \\
&\le \frac{1}{2}(2|\delta_d| + \sum_x |\delta_g(x)|) \le (\kappa+1)\cdot 2^{-\mu}
\end{aligned} \tag{9}$$

Next we consider the statistical distance between $\mathcal{A} = \mathcal{B}(\exp(-u/r))$ and the output distribution $\mathcal{A}'$ of the Algorithm 4. Notice the $\mathcal{A}$ is sampled by combined l independent Bernoulli samples from $\mathcal{A}_i = \mathcal{B}(\exp(-2^i/r))$, such that

$$f_A(1) = \prod_{i=0}^{l-1}\left(1 - u_i f_{\mathcal{A}_i}(0)\right) = \prod_{i=0}^{l-1}(1 - u_i p_i)$$

Here $u = \sum_{i=0}^{l-1} u_i 2^i$ with $u_i \in \{0,1\}$. Then, we have

$$f_{\mathcal{A}'}(1) = \prod_{i=0}^{l-1}(1 - u_i(p_i + \delta_i))$$

for each $|\delta_i| \le 2^{-\mu}$ and $p_i + \delta_i \in [0,1]$. Denote $d_i = 1 - u_i p_i$, then $d_i - u_i\delta_1 \in [0,1]$ and $d_i \in [0,1]$. It leads to

$$\begin{aligned}
&\mathrm{SD}(\mathcal{A}', \mathcal{A}) = |f_{\mathcal{A}'}(1) - f_{\mathcal{A}}(1)| \\
&= \left|\prod_{i=0}^{l-1}(d_i - u_i\delta_i) - \prod_{i=0}^{l-1} d_i\right| \le \sum_{i=0}^{l-1} u_i|\delta_i| \le l\cdot 2^{-\mu}
\end{aligned} \tag{10}$$

Finally, we focus on rejection sampling $\mathcal{N}_{\mathbb{Z},N}$. Actually, we use $\mathcal{L}'$ as proposal distribution and sample accept bit from $\mathcal{A}'$. According to Equation (9), Equation (10), we have

$$\begin{aligned}
f_{\mathcal{L}'}(x) &= f_{\mathcal{L}_{\mathbb{Z},N}}(x) + \delta_l(x) \\
f_{\mathcal{A}'}(1;x) &= f_{\mathcal{A}}(1;x) + \delta_a = \frac{f_{\mathcal{N}_{\mathbb{Z},N}}(x)}{M \cdot f_{\mathcal{L}_{\mathbb{Z},N}}(x)} + \delta_a
\end{aligned}$$

with $\sum_x |\delta_l(x)| \le 2(\kappa+1)\cdot 2^{-\mu}$ and $\delta_a \le l\cdot 2^{-\mu}$. The actual average accept probability p' is:

$$\begin{aligned}
p' &= \sum_x f_{\mathcal{L}'}(x)\cdot f_{\mathcal{A}'}(1;x) \\
&= \sum_x (f_{\mathcal{L}_{\mathbb{Z},N}}(x) + \delta_l(x))\cdot\left(\frac{f_{\mathcal{N}_{\mathbb{Z},N}}(x)}{M\cdot f_{\mathcal{L}_{\mathbb{Z},N}}(x)} + \delta_a\right) \\
&= \frac{1}{M} + \sum_x \delta_n(x) = p_* + \sum_x \delta_n(x)
\end{aligned}$$

Here $\delta_n(x) = \delta_l(x)\cdot f_{\mathcal{A}'}(1;x) + f_{\mathcal{L}_{\mathbb{Z},N}}(x)\cdot\delta_a$, which satisfies

$$\sum_x |\delta_n(x)| \le \sum_x |\delta_l(x)| + |\delta_a| \le (2\kappa + l + 2)\cdot 2^{-\mu}$$

Therefore, the lower bound of p' is

$$p' \ge p_* - \sum_x |\delta_n(x)| \ge p_* - (2\kappa + l + 2)\cdot 2^{-\mu} = p'_*$$

And the actual target distribution $\mathcal{W}$ becomes

$$f_{\mathcal{W}}(x) = \frac{f_{\mathcal{L}'}(x)\cdot f_{\mathcal{A}'}(1;x)}{p'} = \frac{p_* f_{\mathcal{N}_{\mathbb{Z},N}}(x) + \delta_n(x)}{p_* + \sum_t \delta_n(t)}$$

The statistical distance between the $\mathcal{W}$ and $\mathcal{N}_{\mathbb{Z},N}$ is

$$\begin{aligned}
\mathrm{SD}(\mathcal{W}, \mathcal{N}_{\mathbb{Z},N}) &= \frac{1}{2}\sum_x \left|f_{\mathcal{W}}(x) - f_{\mathcal{N}_{\mathbb{Z},N}}(x)\right| \\
&= \frac{1}{2}\sum_x \left|\frac{p_* f_{\mathcal{N}_{\mathbb{Z},N}}(x) + \delta_n(x)}{p_* + \sum_t \delta_n(t)} - f_{\mathcal{N}_{\mathbb{Z},N}}(x)\right| \\
&= \frac{\sum_x |\delta_n(x) - \sum_t \delta_n(t) f_{\mathcal{N}_{\mathbb{Z},N}}(x)|}{2(p_* + \sum_t \delta_n(t))} \\
&\le \frac{\sum_x |\delta_n(x)| + \sum_t |\delta_n(t)|\cdot \sum_x f_{\mathcal{N}_{\mathbb{Z},N}}(x)}{2(p_* + \sum_t \delta_n(t))} \\
&\le \frac{(2\kappa + l + 2)\cdot 2^{-\mu}}{p'} \le \frac{(2\kappa + l + 2)\cdot 2^{-\mu}}{p'_*}
\end{aligned}$$

If follows that

$$\mathrm{SD}(\mathcal{W}^n, \mathcal{N}^n_{\mathbb{Z},N}) \le \frac{n}{p'_*}\cdot(2\kappa + l + 2)\cdot 2^{-\mu} \tag{11}$$

Denote $\mathcal{V}$ as the output distribution of Algorithm 6, which uses m trials to generate n independent samples in $\mathcal{W}$. Denote E as the event that m trials fail to generate n samples, s.t.

$$\begin{aligned}
\Pr[E] &= \Pr\left[X < n \mid X \sim \mathbf{Bin}(m, p')\right] \\
&\le \Pr\left[X < n \mid X \sim \mathbf{Bin}(m, p'_*)\right]
\end{aligned}$$

Here $\mathcal{I}$ is the binomial distribution. Then we have

$$f_{\mathcal{V}}(x) = \Pr[\neg E]\cdot f_{\mathcal{W}^n}(x) + \Pr[E]\cdot e(x)$$

Here $e(x)$ is the probability of output x when E happens. The statistical distance between $\mathcal{W}^n$ and $\mathcal{V}$ is:

$$\begin{aligned}
\mathrm{SD}(\mathcal{V}, \mathcal{W}^n) &= \frac{1}{2}|f_{\mathcal{V}}(x) - f_{\mathcal{W}^n}(x)| \\
&= \frac{1}{2}\sum_x |\Pr[E]\,(f_{\mathcal{W}^n}(x) - e(x))| \\
&\le \frac{1}{2}\cdot\Pr[E]\left(\sum_x f_{\mathcal{W}^n}(x) + \sum_x e(x)\right) = \Pr[E]
\end{aligned} \tag{12}$$

Combine Equation (7), Equation (11), Equation (12), according to the triangle inequality, we obtain that

$$\begin{aligned}
\mathrm{SD}(\mathcal{N}^n_{\mathbb{Z}}, \mathcal{V}) &\le \mathrm{SD}(\mathcal{N}^n_{\mathbb{Z}}, \mathcal{N}^n_{\mathbb{Z},N}) \\
&\quad + \mathrm{SD}(\mathcal{N}^n_{\mathbb{Z},N}, \mathcal{W}^n) + \mathrm{SD}(\mathcal{N}^n_{\mathcal{W}^n}, \mathcal{V}) \\
&= \delta_t + \delta_b + \delta_r
\end{aligned}$$

Here

$$\begin{aligned}
\delta_t &= 2ne^{-\frac{N^2}{2\sigma^2}} \\
\delta_b &= \frac{n}{p_*}\cdot(2\kappa + l + 2)\cdot 2^{-\mu} \\
\delta_r &= \Pr\left[X < n \mid X \sim \mathcal{I}(m, p'_*)\right]
\end{aligned}$$

with $p'_* = p_* - (2\kappa + l + 2)\cdot 2^{-\mu}$