Gaussian Sampling over the Integers: Efficient, Generic, Constant-Time

Daniele Micciancio
UCSD
daniele@cs.ucsd.edu

Michael Walter
UCSD
miwalter@eng.ucsd.edu

July 21, 2017

Abstract

Sampling integers with Gaussian distribution is a fundamental problem that arises in almost every application of lattice cryptography, and it can be both time consuming and challenging to implement. Most previous work has focused on the optimization and implementation of integer Gaussian sampling in the context of specific applications, with fixed sets of parameters. We present new algorithms for discrete Gaussian sampling that are both generic (application independent), efficient, and more easily implemented in constant time without incurring a substantial slow-down, making them more resilient to side-channel (e.g., timing) attacks. As an additional contribution, we present new analytical techniques that can be used to simplify the precision/security evaluation of floating point cryptographic algorithms, and an experimental comparison of our algorithms with previous algorithms from the literature.

1 Introduction

Lattice-based cryptography has gained much popularity in recent years, not only within the cryptographic community, but also in the area of computer security in both research and industry, for at least two reasons: first, many classical cryptographic primitives can be realized very efficiently using lattices, providing strong security guarantees, including conjectured security against quantum computers [14, 4, 39]. Second, lattices allow to build advanced schemes that go beyond classical public key encryption, like fully homomorphic encryption [9, 21, 16, 8], identity based encryption [2, 1], attribute based encryption [6, 7], some forms of multilinear maps [20, 26] and even some forms of program obfuscation [10]. Discrete Gaussian distributions (i.e., normal Gaussian distributions on the real line, but restricted to take integer values), play a fundamental role in lattice cryptography: Gaussian sampling is at the core of security proofs (from worst-case lattice problems) supporting both the conjectured hardness of the Learning With Errors (LWE) problem [42, 41, 27, 36], and the tightest reductions for the Short Integer Solution (SIS) problem [30, 29], which provide a theoretical foundation to the field. The use of Gaussian distributions is especially important in the context of the most advanced cryptographic applications of lattices that make use of preimage sampling [22, 37, 28], as the use of other distributions can easily leak information about secret keys and open cryptographic primitives to devastating attacks [34]. Even in the technically simpler context of LWE noise generation, where Gaussian distributions can be safely replaced by more easily samplable (e.g., uniform) distributions (see e.g. [12, 29]), this requires a noticeable increase in the noise level, resulting in substantial performance degradation, and still points to discrete Gaussian distributions as the most desirable choice to achieve good performance/security trade-offs. In summary, despite continued theoretical efforts and practical attempts to replace Gaussian distributions with more implementation friendly ones, and a few isolated examples where discrete Gaussians can be avoided altogether with almost no penalty [4], the cryptography research community has been converging to accept discrete Gaussian sampling as one of the fundamental building blocks of lattice cryptography.

Gaussian sampling aside, lattice cryptography can be very attractive from an implementation standpoint, requiring only simple arithmetic operations on small integer numbers (easily fitting a computer word on commodity microprocessors), and offering ample opportunities for parallelization at the register and processor level, both in hardware and in software implementations. In this respect, discrete Gaussian sampling can often be the main hurdle in implementation/optimization efforts, and a serious bottleneck to achieve good performance in practice. As many primitives find their way into practical implementations [3, 15] and lattice cryptography is considered for possible standardization as a post-quantum security solution [35], the practical aspects of discrete Gaussian sampling (including efficiency, time-memory trade-offs, side-channel resistance, etc.) have started to attract the attention of the research community, e.g., see [11, 43, 14, 13, 33, 18, 46, 17]. However, most of these works address the problem of Gaussian sampling in the context of a specific application, and for specific values of the parameters and settings that come to define the discrete Gaussian sampling problem: the standard deviation of the Gaussian distribution, the center (mean) of the Gaussian, how these values depend on the targeted security level, and whether the values are fixed once and for all, or during key generation time, or even on a sample-by-sample basis. So, while implementation efforts have clearly demonstrated that (if properly specialized and optimized) discrete Gaussian sampling can be used in practice, it is unclear to what extent optimized solutions can be ported from one application to another, and even when this is possible, achieving good performance still seems to require a disproportionate amount of effort. Finally, achieving security against side-channel (e.g., timing) attacks has been recognized as an important problem [23, 38, 45], but developing constant-time implementations of Gaussian sampling without incurring major performance penalties is still a largely unsolved problem.

Our Contribution. We develop of a new discrete Gaussian sampling algorithm over the integers with a unique set of desirable properties that make it very attractive in cryptographic applications. The new algorithm

- can be used to sample efficiently from discrete Gaussian distributions with arbitrary and varying parameters (standard deviation and center), enabling its use in a wide range of applications.
- provides a time-memory trade-off, the first of its kind for sampling with varying parameters, allowing to fine-tune the performance on different platforms.
- can be split into an offline and online phase, where the offline phase can be carried out even before knowing the parameters of the requested distribution. Moreover, both phases can be implemented in constant time with only minor performance degradation, providing resilience against timing side-channel attacks.
- can be parallelized and optimized, both in hardware and software, in a largely application-independent manner.

We demonstrate the efficiency of the new algorithm both through a rigorous theoretical analysis, and practical experimentation with a prototype implementation. Our experimental results show that our new algorithms achieve generality and flexibility without sacrificing performance, matching, or even beating the online phase of previous (specialized) algorithms. See next paragraph and Sect. 6.6 for details.

A recurring problem in the analysis of Gaussian sampling (or other probabilistic algorithms involving the use of real numbers at some level), is to accurately account for how the use of floating point approximations affects performance and security. This is often a critical issue in practice, as using standard (53 bit) double precision floating point numbers offers major efficiency advantages over the use of arbitrary precision arithmetic libraries, but can have serious security implications when targeting 80bit or 100bit security levels. As an additional contribution, we develop new analytical tools for the accuracy/security analysis of floating point algorithms, and exemplify their use in the analysis of our new Gaussian sampling algorithm. More specifically, we propose a new notion of closeness between probability distributions (which we call the "max-log" distance), that combines the simplicity and ease of use of statistical distance (most commonly used in cryptography), with the effectiveness of Rényi and KL divergences recently used in cryptography to

obtain sharp security estimates [39, 5, 40]. The new measure is closely related to the standard notion of relative error and the Rényi divergence of order ∞, but it is easier to define[1] and it is also a metric, i.e., it enjoys the (symmetric and triangle inequality) properties that make the statistical distance a convenient tool for the analysis of complex algorithms. Using this new metric, we show that our new algorithms can be implemented using standard (extended) double precision floating point arithmetic, and still provide a more than adequate (100 bits or higher) level of security.

Finally, we also evaluate different algorithms for discrete Gaussian sampling experimentally in a common setting. While previous surveys [19] and experimental studies [11, 24] exist, they either do not provide a fair comparison or are incomplete. Somewhat surprisingly, an algorithm [25] that has gone mostly unnoticed in the cryptographic community so far, emerged as very competitive solution in our study, within the class of variable-time algorithms that can be used when timing attacks are not a concern.

Techniques. The main idea behind our algorithm is to reduce the general discrete Gaussian sampling problem (for arbitrary standard deviation s and center c), to the generation (and recombination) of a relatively small number of samples coming from a Gaussian distribution for a fixed and rather small value of s. Reducing the general problem to discrete Gaussian sampling for a fixed small value of s has several advantages:

- Gaussian sampling for fixed parameters can be performed more efficiently than general Gaussian sampling because the probability tables or tree traversal data structures required by the basic sampler can be precomputed. Moreover, as the standard deviation s of the basic sampler is small, these tables or data structures only require a very modest amount of memory.
- Since the parameters of the basic sampler are fixed and do not depend on the application input, the basic samples can be generated offline. The online (recombination) phase of the algorithm is very fast, as it only needs to combine a small number of basic samples.
- The online (recombination) phase of the algorithm is easily implemented in constant time, as the number of operations it performs only depends on the application parameters, and not on the actual input values or randomness. The offline phase can also be made constant time with only a minor performance penalty, observing that basic samples are always generated and used in batches. So, instead of requiring the generation of each basic sample to take a fixed amount of time, one can look at the time to generate a batch of samples in the aggregate. Since the basic samples are totally independent, their aggregate generation time is very sharply concentrated around the expectation, and can be made constant (except with negligible probability) simply by adding a small time penalty to the generation of the whole batch.
- The parameters of the basic sampler are fixed once and for all, and do not depend on the parameters of online phase and final application. This opens up the possibility of a hybrid hardware/software implementation, where the basic sampler is optimized and implemented once and for all, perhaps in hardware, and making efficient use of parallelism. The fast recombination phase is quickly executed in software by combining the samples generated by the hardware module, based on the application parameters.

The method we use to combine the basic samples extends and generalizes techniques that have been used in the implementation of Gaussian samplers before. The work most closely related to ours is [39], which generates Gaussian samples with a relatively large standard deviation s by first computing two samples x_1, x_2 with smaller standard deviation $\approx \sqrt{s}$, and then computing $kx_1 + x_2$, for $k \approx \sqrt{s}$. We improve on this basic idea in several dimensions:

- First, we use the idea recursively, obtaining x_1 and x_2 also by combining multiple samples with even smaller standard deviation. While recursion is a rather natural and simple idea, and it was already

[1] The distance between two discrete distributions $\mathcal{P}$ and $\mathcal{Q}$ (with the same support S), is simply the maximum (over $x \in S$) of $|\log \mathcal{P}(x) - \log \mathcal{Q}(x)|$.

mentioned in [39], the realization that the performance benefits of using basic samples with even smaller standard deviation more than compensate the overhead associated to computing several samples is new.

- Second, we employ a convolution theorem from [29] to combine the samples (at each level of the recursion). This allows for greater flexibility in the choice of parameters, for example the number of samples to combine at each level or the choice of coefficients. This can be important in the context of side-channel attacks as demonstrated in [38].

- Finally, we generalize the algorithm to sample according to Gaussian distributions with arbitrary center as follows. Assume the center c has k binary fractional digits, i.e., $c \in \mathbb{Z}/2^k$. Then, we can use a first integer Gaussian sample (scaled by a factor 2^{-k}) to randomly round c to a center in $\mathbb{Z}/2^{k-1}$. Then, we use a second sample (scaled by $2^{-(k-1)}$) to round the new center to a coarser set $\mathbb{Z}/2^{k-2}$, and so on for k times, until we obtain a sample in $\mathbb{Z}$ as desired. Since the final output is obtained by combining a number of Gaussian samples together, the result still follows a discrete Gaussian distribution. Moreover, since the scaling factors grow geometrically, the standard deviation of the final output is (up to a small constant factor) the same as the one of the original samples.

The algorithms presented in this paper include several additional improvements and optimizations, as described below. Using different values for the standard deviation of the basic sampler, and expressing the center of the Gaussian c to a base other than 2, allows various time-memory trade-offs that can be used to fine-tune the performance of the algorithm to different platforms. The exact value of the standard deviation of the final output distribution can be finely adjusted by adding some noise to the initial center and invoking the convolution theorem of [37]. Finally, when the center of the Gaussian c is a high precision floating point number, the number of iterations (and basic samples required) can be greatly reduced by first rounding it to a coarser grid using a simple biased coin flip, and using our *max-log* metric to get sharper estimates on the number of precision bits required.

Outline. We begin by introducing some notation in Sect. 2, and a general framework for the analysis of approximate samplers in Sect. 3. In Sect. 4 we introduce our new "max-log" metric, which we will use to simplify the analysis for complex sampling algorithms. Our new sampling algorithms are presented in Sect. 5. Section 6 concludes the paper with a description of our experimental results. Appendix A discusses pitfalls in a previous work [44] attempting to tighten the precision/security analysis of approximate samplers.

2 Preliminaries

Notation. We denote the integers by $\mathbb{Z}$ and the reals by $\mathbb{R}$. Roman and Greek letters can denote elements from either set, while bold letters denote vectors over them. Occasionally, we construct vectors on the fly using the notation $(\cdot)_{i \in S}$ for some set S (or in short $(\cdot)_i$ if the set S is clear from context), where $\cdot$ is a function of i. We denote the logarithm with base 2 by log and the one with base e by ln.

Calligraphic letters are reserved for probability distributions and $x \leftarrow \mathcal{P}$ means that x is sampled from the distribution $\mathcal{P}$. For any x in the support of $\mathcal{P}$ we denote its probability under $\mathcal{P}$ by $\mathcal{P}(x)$. All distributions in this work are discrete. The statistical distance between two distributions $\mathcal{P}$ and $\mathcal{Q}$ over the same support S is defined as $\Delta_{\text{SD}}(\mathcal{P}, \mathcal{Q}) = \frac{1}{2} \sum_{x \in S} |\mathcal{P}(x) - \mathcal{Q}(x)|$ and the KL-divergence as $\delta_{\text{KL}}(\mathcal{P}, \mathcal{Q}) = \sum_{x \in S} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}$. Note that the former is a metric, while the latter is not. Pinsker's inequality bounds Δ_{SD} in terms of δ_{KL} by $\Delta_{\text{SD}}(\mathcal{P}, \mathcal{Q}) \leq \sqrt{\delta_{\text{KL}}(\mathcal{P}, \mathcal{Q})/2}$. A probability ensemble $\mathcal{P}_\theta$ is a family of distributions indexed by a parameter θ (which is possibly a vector). We extend any measure δ between distributions to probability ensembles as $\delta(\mathcal{P}_\theta, \mathcal{Q}_\theta) = \max_\theta \delta(\mathcal{P}_\theta, \mathcal{Q}_\theta)$. For notational simplicity, we do not make a distinction between random variables, probability distributions, and probabilistic algorithms generating them. An algorithm A with oracle access to a sampler for distribution ensemble $\mathcal{P}_\theta$ is denoted by $A^{\mathcal{P}}$, which means that it adaptively sends queries θ_i to the sampler, which returns a sample from $\mathcal{P}_{\theta_i}$. If A uses only one sample from $\mathcal{P}_\theta$, then we write $A(\mathcal{P}_\theta)$.

In this work we will occasionally encounter expressions of the form $\epsilon + O(\epsilon^2)$ for some small ϵ. In all of these cases, the constant c hidden in the asymptotic notation is much smaller than $1/\epsilon$ (say $c\epsilon \leq 2^{-30}$). So, the higher order term $O(\epsilon^2)$ has virtually no impact, neither in practice nor asymptotically, on our applications. We define $\hat{\epsilon} = \epsilon + O(\epsilon^2)$ and write $a \simeq b$ for $a = \hat{b}$, and similarly $a \lesssim b$ for $a \leq \hat{b}$. This allows us to drop the $O(\epsilon^2)$ term and avoid tracing irrelevant terms through our calculations without losing rigor, e.g. $\ln(1+\epsilon) = \epsilon + O(\epsilon^2)$ can be written as $\ln(1+\epsilon) \simeq \epsilon$.

For $c \in [0, 1)$ and $k \in \mathbb{Z}$ we define rounding operators $\lceil c \rceil_k = \lceil 2^k c \rceil / 2^k$ and $\lfloor c \rfloor_k = \lfloor 2^k c \rfloor / 2^k$, which round c (up or down, respectively) to a number with k fractional bits. We also define a randomized rounding operator $\lfloor c \rceil_k = \lfloor c \rfloor_k + \mathcal{B}_\alpha / 2^k$ (where $\mathcal{B}_\alpha$ is a Bernoulli random variable of parameter $\alpha = 2^k c \mod 1$) which rounds c to either $\lceil c \rceil_k$ (with probability α) or $\lfloor c \rfloor_k$ (with probability $1 - \alpha$).

Approximations of Real Numbers. A p-bit floating point (FP) approximation $\bar{x}$ of a real x stores the p most significant bits of x together with a binary exponent. This guarantees that the relative error is bounded by $\delta_{\mathrm{RE}}(x, \bar{x}) = |x - \bar{x}|/|x| \leq 2^{-p}$. We extend the notion of relative error to any two distributions $\mathcal{P}$ and $\mathcal{Q}$

$$\delta_{\mathrm{RE}}(\mathcal{P}, \mathcal{Q}) = \max_{x \in S} \delta_{\mathrm{RE}}(\mathcal{P}(x), \mathcal{Q}(x)) = \max_{x \in S} \frac{|\mathcal{P}(x) - \mathcal{Q}(x)|}{\mathcal{P}(x)},$$

where S is the support of $\mathcal{P}$. It is straightforward to verify that $\Delta_{\mathrm{SD}}(\mathcal{P}, \mathcal{Q}) \leq \frac{1}{2}\delta_{\mathrm{RE}}(\mathcal{P}, \mathcal{Q})$. The relative error can also be used to bound the KL-divergence:

Lemma 2.1 (Strengthening [39, Lemma 2]) *For any two distributions $\mathcal{P}$ and $\mathcal{Q}$ with the same support S with $\mu = \delta_{\mathrm{RE}}(\mathcal{P}, \mathcal{Q}) < 1$,*

$$\delta_{\mathrm{KL}}(\mathcal{P}, \mathcal{Q}) \leq \frac{\mu^2}{2(1-\mu)^2}.$$

In particular, if $\mu \leq 1/4$, then $\delta_{\mathrm{KL}}(\mathcal{P}, \mathcal{Q}) \leq (8/9)\mu^2 < \mu^2$.

Proof: Recall that $\delta_{\mathrm{KL}}(\mathcal{P}, \mathcal{Q}) = \sum_i \mathcal{P}(i) \ln(\mathcal{P}(i)/\mathcal{Q}(i))$. For any $p, q > 0$, let $x = (p-q)/p = 1 - (q/p) < 1$, so that $\ln(p/q) = -\ln(1-x) = x + e(x)$ with error function $e(x) = -x - \ln(1-x)$. Notice that $e(0) = 0$, $e'(0) = 0$ and $e''(x) = 1/(1-x)^2 \leq 1/(1-\mu)^2$ for all $x \leq \mu$. It follows that $e(x) \leq x^2/(2(1-\mu)^2) \leq \mu^2/(2(1-\mu)^2)$ for all $|x| \leq \mu$, and

$$\delta_{\mathrm{KL}}(\mathcal{P}, \mathcal{Q}) = \sum_i \mathcal{P}(i) \ln\left(\frac{\mathcal{P}(i)}{\mathcal{Q}(i)}\right) \leq \sum_i \mathcal{P}(i) \cdot \left(\frac{\mathcal{P}(i) - \mathcal{Q}(i)}{\mathcal{P}(i)} + e\right) = 1 - 1 + e = e$$

where $e = \mu^2/(2(1-\mu)^2)$.□ This is a slight improvement over [39, Lemma 2], which shows that if $\mu \leq 1/4$, then $\delta_{\mathrm{KL}}(\mathcal{P}, \mathcal{Q}) \leq 2\mu^2$. So, Lemma 2.1 improves the bound by a constant factor $9/4$. In fact, for $\mu \approx 0$, Lemma 2.1 shows that the bound can be further improved to $\delta_{\mathrm{KL}}(\mathcal{P}, \mathcal{Q}) \lesssim \frac{1}{2}\mu^2$.

Discrete Gaussians. Let $\rho(x) = \exp(-\pi x^2)$ be the Gaussian function with total mass $\int_x \rho(x) = 1$. We extend it to countable sets A by $\rho(A) = \sum_{x \in A} \rho(x)$. We write $\rho_{c,s}(x) = \rho((x-c)/s)$ for the Gaussian function centered around c and scaled by a factor s. The discrete Gaussian distribution over the integers, denoted $\mathcal{D}_{\mathbb{Z},c,s}$, is the distribution that samples $y \leftarrow \mathcal{D}_{\mathbb{Z},c,s}$ with probability $\rho_{c,s}(y)/\rho_{c,s}(\mathbb{Z})$ for any $y \in \mathbb{Z}$. Sampling from $\mathcal{D}_{\mathbb{Z},c,s}$ is computationally equivalent to sampling from $\mathcal{D}_{c+\mathbb{Z},s}$, the centered discrete Gaussian over the coset $c + \mathbb{Z}$. For any $\epsilon > 0$, the smoothing parameter [30] of the integers $\eta_\epsilon(\mathbb{Z})$ is the smallest $s > 0$ such that $\rho(s\mathbb{Z}) \leq 1 + \epsilon$. A special case of [30, Lemma 3.3] shows that the smoothing parameter satisfies

$$\eta_\epsilon(\mathbb{Z}) \leq \sqrt{\ln(2 + 2/\epsilon)/\pi}.$$

So, $\eta_\epsilon(\mathbb{Z}) < 6$ is a relatively small constant even for very small values of $\epsilon < 2^{-160}$. Another useful bound, which easily follows from Poisson summation formula [30, Lemma 2.8], is $\delta_{\mathrm{RE}}(s, \rho_{c,s}(\mathbb{Z})) \leq \delta_{\mathrm{RE}}(s, \rho_s(\mathbb{Z})) = \rho(s\mathbb{Z}) - 1$. Therefore, for any $s \geq \eta_\epsilon(\mathbb{Z})$, and $c \in \mathbb{R}$, we have

$$\delta_{\mathrm{RE}}(s, \rho_{c,s}(\mathbb{Z})) \leq \epsilon,$$

i.e., the total measure of $\rho_{c,s}(\mathbb{Z})$ approximates s. We will use the smoothing parameter to invoke the following tail bound and discrete convolution theorems.

Lemma 2.2 ([22, Lemma 4.2 (ePrint)]) *For any $\epsilon > 0$, any $s > \eta_\epsilon(\mathbb{Z})$, and any $t > 0$,*

$$\Pr_{x \leftarrow \mathcal{D}_{\mathbb{Z},c,s}}[|x - c| \geq t \cdot s] \leq 2e^{-\pi t^2} \cdot \frac{1+\epsilon}{1-\epsilon}.$$

Theorem 2.1 ([29, Theorem 3]) *Let Λ be an n-dimensional lattice, $\mathbf{z} \in \mathbb{Z}^m$ a nonzero integer vector, $\mathbf{s} \in \mathbb{R}^m$ with $s_i \geq \sqrt{2}\|\mathbf{z}\|_\infty \eta_\epsilon(\mathbb{Z})$ for all $i \leq m$ and $\mathbf{c}_i + \Lambda$ arbitrary cosets. Let $\mathbf{y}_i$ be independent samples from $\mathcal{D}_{\mathbf{c}_i+\Lambda,s_i}$, respectively. Then the distribution of $\mathbf{y} = \sum z_i \mathbf{y}_i$ is close to $\mathcal{D}_{Y,s}$, where $Y = \sum_i z_i \mathbf{c}_i + \gcd(\mathbf{z})\Lambda$ and $s = \sqrt{\sum_i z_i^2 s_i^2}$. In particular, if $\tilde{\mathcal{D}}_{Y,s}$ is the distribution of $\mathbf{y}$, then $\delta_{\mathrm{RE}}(\mathcal{D}_{Y,s}, \tilde{\mathcal{D}}_{Y,s}) \leq \frac{1+\epsilon}{1-\epsilon} - 1 \simeq 2\epsilon$.*

The theorem is stated in its full generality, but in this work we will only use it for the one dimensional lattice $\mathbb{Z}$ and for the case that $\mathbf{c}_i = 0$ and $\gcd(\mathbf{z}) = 1$.

Theorem 2.2 ([37, Theorem 1]) *Let $\mathbf{S}_1, \mathbf{S}_2 > \mathbf{0}$ be positive definite matrices, with $\mathbf{S} = \mathbf{S}_1 + \mathbf{S}_2$ and $\mathbf{S}_3^{-1} = \mathbf{S}_1^{-1} + \mathbf{S}_2^{-1} > \mathbf{0}$. Let Λ_1, Λ_2 be lattices such that $\sqrt{\mathbf{S}_1} \geq \eta_\epsilon(\Lambda_1)$ and $\sqrt{\mathbf{S}_3} \geq \eta_\epsilon(\Lambda_2)$ for some positive $\epsilon \leq 1/2$, and let $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{R}^n$ be arbitrary. Then the distribution of $\mathbf{x}_1 \leftarrow \mathbf{x}_2 + \mathcal{D}_{\mathbf{c}_1 - \mathbf{x}_2 + \Lambda_1, \sqrt{\mathbf{S}_1}}$, where $\mathbf{x}_2 \leftarrow \mathcal{D}_{\mathbf{c}_2+\Lambda_2,\sqrt{\mathbf{S}_2}}$, is close to $\mathcal{D}_{c_1+\Lambda_1,\sqrt{\mathbf{S}_1}}$. In particular, if $\tilde{\mathcal{D}}_{c_1+\Lambda_1,\sqrt{\mathbf{S}_1}}$ is the distribution of $\mathbf{x}_1$, then $\delta_{\mathrm{RE}}(\mathcal{D}_{c_1+\Lambda_1,\sqrt{\mathbf{S}_1}}, \tilde{\mathcal{D}}_{c_1+\Lambda_1,\sqrt{\mathbf{S}_1}}) \leq \left(\frac{1+\epsilon}{1-\epsilon}\right)^2 - 1 \simeq 4\epsilon$.*

Again, we stated the theorem in its full generality, but we will only need it for one dimensional lattices. Accordingly, $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$ will simply be (the square of) real noise parameters s_1, s_2, s_3.

3 The Security of Approximate Samplers

Many security reductions for lattice-based cryptographic primitives assume that the primitive has access to samplers for an ideal distribution, which may be too difficult or costly to sample from, and is routinely replaced by an approximation in any concrete implementation. Naturally, if the approximation is good enough, then security with respect to the ideal distribution implies that the actual implementation (using the approximate distribution) is also secure. But evaluating how the quality of approximation directly affects the concrete security level achieved by the primitive can be a rather technical task. Traditionally, the quality of the approximation has been measured in terms of the statistical distance $\delta = \Delta_{\mathrm{SD}}$, which satisfies the following useful properties:

1. *Probability preservation:* for any event E over the random variable X we have $Pr_{X \leftarrow \mathcal{P}}[E] \geq Pr_{X \leftarrow \mathcal{Q}}[E] - \delta(\mathcal{P}, \mathcal{Q})$. This property allows to bound the probability of an event occurring under $\mathcal{P}$ in terms of the probability of the same event occurring under $\mathcal{Q}$ and the quantity $\delta(\mathcal{P}, \mathcal{Q})$. It is easy to see that this property is equivalent to the bound $\Delta_{\mathrm{SD}}(\mathcal{P}, \mathcal{Q}) \leq \delta(\mathcal{P}, \mathcal{Q})$. So the statistical distance $\delta = \Delta_{\mathrm{SD}}$ satisfies this property by definition.

2. *Sub-additivity for joint distributions:* if $(X_i)_i$ and $(Y_i)_i$ are two lists of discrete random variables over the support $\prod_i S_i$, then

$$\delta((X_i)_i, (Y_i)_i) \leq \sum_i \max_a \delta([X_i \mid X_{<i} = a], [Y_i \mid Y_{<i} = a]),$$

   where $X_{<i} = (X_1, \ldots, X_{i-1})$ (and similarly for $Y_{<i}$), and the maximum is taken over $a \in \prod_{j<i} S_j$.

3. *Data processing inequality:* $\delta(f(\mathcal{P}), f(\mathcal{Q})) \leq \delta(\mathcal{P}, \mathcal{Q})$ for any two distributions $\mathcal{P}$ and $\mathcal{Q}$ and (possibly randomized) algorithm $f(\cdot)$, i.e., the measure does not increase under function application.

We call any measure that satisfies these three properties a *useful* measure. Indeed, the usefulness of the measure is illustrated by the following simple lemma, but first we need to define the class of generic cryptographic schemes it applies to.

Definition 1 (Standard cryptographic scheme) *We consider an arbitrary cryptographic scheme S, consisting of one or more algorithms with oracle access to a probability distribution ensemble $\mathcal{P}_\theta$, and whose security against an adversary A (also consisting of one or more algorithms) is described in terms of a game $G^{\mathcal{P}}_{S,A}$ defining the event that A succeeded in breaking the scheme S. The success probability of A against S (when using samples from $\mathcal{P}_\theta$) is defined as $\epsilon^{\mathcal{P}}_A = \Pr\{G^{\mathcal{P}}_{S,A}\}$. The cost of an attack A against S is defined as $t_A/\epsilon^{\mathcal{P}}_A$, and the bit-security of S is the minimum (over all adversaries A) of $\log(t_A/\epsilon^{\mathcal{P}}_A)$.*

For simplicity, we assume that the running time t_A of the game $G^{\mathcal{P}}_{S,A}$ does not depend on the distributions $\mathcal{P}_\theta$, and that the number of calls to $\mathcal{P}_\theta$ performed during any run of the game $G^{\mathcal{P}}_{S,A}$ is bounded from above by t_A.

Lemma 3.1 *Let $S^{\mathcal{P}}$ be a standard cryptographic scheme as in Definition 1 with black-box access to a probability distribution ensemble $\mathcal{P}_\theta$, and δ any cryptographically useful measure. If $S^{\mathcal{P}}$ is κ-bit secure and $\delta(\mathcal{P}_\theta, \mathcal{Q}_\theta) \leq 2^{-\kappa}$, then $S^{\mathcal{Q}}$ is $(\kappa-1)$-bit secure.*

Before we prove Lemma 3.1, we begin with a technical observation that will be used throughout this section.

Lemma 3.2 *Let δ be a cryptographically useful measure, let $\mathcal{P}_\theta$ and $\mathcal{Q}_\theta$ be two probability ensembles and let $A^{\mathcal{P}}$ be an algorithm querying $\mathcal{P}_\theta$ at most q times. Let θ_i (resp. $\tilde{\theta}_i$) be the distribution of the i-th query made by $A^{\mathcal{P}}$ (resp. $A^{\mathcal{Q}}$). Then,*

$$\delta((\theta_i, \mathcal{P}_{\theta_i} \mid X_i), (\tilde{\theta}_i, \mathcal{Q}_{\tilde{\theta}_i} \mid X_i)) \leq \delta(\mathcal{P}_\theta, \mathcal{Q}_\theta)$$

for any event X_i that $(\theta_j, \mathcal{P}_{\theta_j})_{j<i}$ and $(\tilde{\theta}_j, \mathcal{Q}_{\tilde{\theta}_j})_{j<i}$ take some specific (and identical) value.

Proof: Note that at any point during the execution of A, conditioned on the event X_i, $A^{\mathcal{P}}$ and $A^{\mathcal{Q}}$ behave identically up to the point they make the ith query. In particular, the conditional distributions $(\theta_i \mid X_i)$ and $(\tilde{\theta}_i \mid X_i)$ are identical and $\delta((\theta_i \mid X_i), (\tilde{\theta}_i \mid X_i)) = 0$. It follows by subadditivity (for joint distributions) that

$$\delta((\theta_i, \mathcal{P}_{\theta_i} \mid X_i), (\tilde{\theta}_i, \mathcal{Q}_{\tilde{\theta}_i} \mid X_i)) \leq \delta((\theta_i \mid X_i), (\tilde{\theta}_i \mid X_i)) + \max_\theta \delta(\mathcal{P}_\theta, \mathcal{Q}_\theta) = \max_\theta \delta(\mathcal{P}_\theta, \mathcal{Q}_\theta).$$

□

Proof:[*of Lemma 3.1*] First observe that the number of queries drawn from $\mathcal{P}_\theta$ ($\mathcal{Q}_\theta$ resp.) is bounded by the running time t_A. For arbitrary adversary A, we have $\epsilon^{\mathcal{P}}_A \geq \epsilon^{\mathcal{Q}}_A - \delta(G^{\mathcal{P}}_{S,A}, G^{\mathcal{Q}}_{S,A})$ by probability preservation of δ. By the bound on the number of queries and the data processing inequality, we have $\delta(G^{\mathcal{P}}_{S,A}, G^{\mathcal{Q}}_{S,A}) \leq \delta((\theta_i, \mathcal{P}_{\theta_i})_{i<t_A}, (\tilde{\theta}_i, \mathcal{Q}_{\tilde{\theta}_i})_{i<t_A})$, where $(\theta_i)_{i<t_A}$ (resp. $(\tilde{\theta}_i)_{i<t_A}$) is the sequence of queries made to $\mathcal{P}_\theta$ (resp. $\mathcal{Q}_\theta$) when A is attacking $S^{\mathcal{P}}$ (resp. $S^{\mathcal{Q}}$). By sub-additivity for joint distributions and Lemma 3.2, this quantity is at most $t_A\delta(\mathcal{P}_\theta, \mathcal{Q}_\theta)$. So $\epsilon^{\mathcal{P}}_A \geq \epsilon^{\mathcal{Q}}_A - t_A\delta(\mathcal{P}_\theta, \mathcal{Q}_\theta)$. Dividing by t_A, we get $\frac{\epsilon^{\mathcal{P}}_A}{t_A} \geq \frac{\epsilon^{\mathcal{Q}}_A}{t_A} - \delta(\mathcal{P}_\theta, \mathcal{Q}_\theta) \geq \frac{\epsilon^{\mathcal{Q}}_A}{t_A} - 2^{-\kappa}$. Because $S^{\mathcal{P}}$ is κ bit secure, we have $2^{-\kappa} \geq \frac{\epsilon^{\mathcal{P}}_A}{t_A} \geq \frac{\epsilon^{\mathcal{Q}}_A}{t_A} - 2^{-\kappa}$, or, equivalently, $2^{1-\kappa} \geq \frac{\epsilon^{\mathcal{Q}}_A}{t_A}$. This shows that $\log \frac{t_A}{\epsilon^{\mathcal{Q}}_A} \geq \kappa - 1$, i.e., $S^{\mathcal{Q}}$ provides $\kappa - 1$ bits of security. □

Lemma 3.1 captures the intuition that security with respect to an ideal distribution implies security with respect to any sufficiently good approximation, and it also gives a way to establish concrete security bounds. In order to (almost) preserve κ bits of security, one needs $\delta(\mathcal{P}_\theta, \mathcal{Q}_\theta) < 2^{-\kappa}$, e.g., as obtained, using $\delta = \Delta_{\text{SD}}$ and estimating the ideal probabilities $\mathcal{Q}(x)$ with κ-bit (fixed point or floating point) approximations. Additionally, Lemma 3.1allows us to view $\mathcal{D}_{\mathbb{Z},c,s}$ as a ts-bounded distribution without losing security. Notice that for a security parameter κ we can set t to about $\sqrt{\kappa \ln 2/\pi} \approx \eta_{2^{-\kappa}}(\mathbb{Z})$, which by Lemma 2.2 implies a

statistical distance of less than $2^{-\kappa}$ if $s \geq \eta_\epsilon(\mathbb{Z})$. So in the rest of this work we will identify the unbounded Gaussian distribution $\mathcal{D}_{\mathbb{Z},c,s}$ with its truncation with support $\mathbb{Z} \cap [c \pm ts]$ whenever appropriate.

While using Δ_{SD} is asymptotically efficient, it has been observed that in practice it can lead to unnecessarily large memory cost and slow computations. The work of [39] showed that we can improve the security analysis of approximate distributions. Assume we have a measure δ that satisfies the following strengthening of the probability preservation property:

1.* *Pythagorean probability preservation* with parameter $\lambda \in \mathbb{R}$, which states that for any joint distributions $(\mathcal{P}_i)_i$ and $(\mathcal{Q}_i)_i$ over support $\prod_i S_i$, if

$$\delta(\mathcal{P}_i \mid a_i, \mathcal{Q}_i \mid a_i) \leq \lambda$$

for all i and $a_i \in \prod_{j<i} S_j$, then

$$\Delta_{\text{SD}}((\mathcal{P}_i)_i, (\mathcal{Q}_i)_i) \leq \|(\max_{a_i} \delta(\mathcal{P}_i \mid a_i, \mathcal{Q}_i \mid a_i))_i\|_2.$$

We call a measure that satisfies this property λ-*pythagorean*. A pythagorean measure additionally satisfying sub-additivity for joint distributions and the data processing inequality (i.e. properties 2 and 3) will be called λ-*efficient*. Using a pythagorean δ, we can improve Lemma 3.1as follows.

Lemma 3.3 *Let $S^{\mathcal{P}}$ be a standard cryptographic scheme as in Definition 1 with black-box access to a probability distribution ensemble $\mathcal{P}_\theta$. If $S^{\mathcal{P}}$ is κ-bit secure and $\delta(\mathcal{P}_\theta, \mathcal{Q}_\theta) \leq 2^{-\kappa/2}$ for some $2^{-\kappa/2}$-*efficient *measure δ, then $S^{\mathcal{Q}}$ is $(\kappa-3)$-bit secure.*

Proof: Towards a contradiction, assume for some adversary A we have $\frac{t_A}{\epsilon^{\mathcal{P}}_A} \geq 2^\kappa$, but $\frac{t_A}{\epsilon^{\mathcal{Q}}_A} < 2^{\kappa-3}$. Consider the hypothetical game $[G^{\mathcal{Q}}_{S,A}]^n$ (resp. $[G^{\mathcal{P}}_{S,A}]^n$) consisting of n independent copies of $G^{\mathcal{Q}}_{S,A}$ (resp. $G^{\mathcal{P}}_{S,A}$). Denote the probability of the event that A wins at least one of the n games by $\epsilon^{\mathcal{Q}}_{A^n}$ (resp. $\epsilon^{\mathcal{P}}_{A^n}$). We begin by showing that we can bound $\epsilon^{\mathcal{P}}_{A^n}$ from below in terms of $\epsilon^{\mathcal{Q}}_{A^n}$ using probability preservation and data processing inequality of Δ_{SD}:

$$\epsilon^{\mathcal{P}}_{A^n} \geq \epsilon^{\mathcal{Q}}_{A^n} - \Delta_{SD}([G^{\mathcal{P}}_{S,A}]^n, [G^{\mathcal{Q}}_{S,A}]^n) \geq \epsilon^{\mathcal{Q}}_{A^n} - \Delta_{SD}((\theta_i, \mathcal{P}_{\theta_i})_i, (\tilde{\theta}_i, \mathcal{Q}_{\tilde{\theta}_i})_i)$$

where $(\theta_i)_i$ (resp. $(\tilde{\theta}_i)_i$) is the sequence of queries made during the game $[G^{\mathcal{P}}_{S,A}]^n$ (resp. $[G^{\mathcal{Q}}_{S,A}]^n$). Lemma 3.2 ensures that we can apply pythagorean probability preservation (Property 1*) to obtain

$$\epsilon^{\mathcal{P}}_{A^n} \geq \epsilon^{\mathcal{Q}}_{A^n} - \sqrt{t_{A^n}} \cdot \delta(\mathcal{P}_\theta, \mathcal{Q}_\theta) \geq \epsilon^{\mathcal{Q}}_{A^n} - \sqrt{t_{A^n}} \cdot 2^{-\kappa/2} \geq \epsilon^{\mathcal{Q}}_{A^n} - \sqrt{\frac{n \cdot t_A}{2^\kappa}}. \tag{1}$$

Now we set $n = 1/\epsilon^{\mathcal{Q}}_A$ so that $\epsilon^{\mathcal{Q}}_{A^n} = 1 - (1-\epsilon^{\mathcal{Q}}_A)^n > 1 - \exp(-1)$. Substituting into (1) and using $\frac{t_A}{\epsilon^{\mathcal{Q}}_A} < 2^{\kappa-3}$ we get

$$\epsilon^{\mathcal{P}}_{A^n} > 1 - \exp(-1) - \sqrt{\frac{t_A}{2^\kappa \epsilon^{\mathcal{Q}}_A}} > 1 - \exp(-1) - 2^{-3/2} \approx 0.279.$$

Finally, to achieve a contradiction, we derive a simple upper bound. By union bound $\epsilon^{\mathcal{P}}_{A^n} \leq n\epsilon^{\mathcal{P}}_A$. Since $S^{\mathcal{P}}$ is κ-bit secure, $\epsilon^{\mathcal{P}}_A \leq t_A/2^\kappa$, which shows that

$$\epsilon^{\mathcal{P}}_{A^n} \leq \frac{nt_A}{2^\kappa} = \frac{t_A}{2^\kappa \epsilon^{\mathcal{Q}}_A} < 2^{-3} = 0.125$$

which is smaller than the lower bound. □

This shows that $\delta(\mathcal{P}_\theta, \mathcal{Q}_\theta) \sim 2^{-\kappa/2}$ is sufficient to maintain κ bits of security. This type of analysis was first used in [39] for the special case of fixed distributions (i.e. θ is fixed and cannot be chosen by the

adversary) and the KL-divergence $\delta = \sqrt{\delta_{\mathrm{KL}}}$, which is efficient (see e.g. [5, 39] for proofs). Lemma 2.1, in combination with Lemma 3.3, shows that it is sufficient for algorithms to approximate the probabilities of the target distribution with floating point numbers of precision about half the security parameter. Interestingly, in this setting, it is important to approximate probabilities in floating point, as $\kappa/2$ bits of fixed-point precision is not secure. (See the full version [31] for an attack.)

In this work, we make use of the Theorem 2.1 and 2.2 to reduce the task of generating a specific discrete Gaussian, to generating samples from different distributions. Observe that these theorems assume access to exact samplers. In order to analyze our algorithms, we need to bound the divergence from the true distribution when applying the theorems to samples from a distribution close to the exact Gaussian distributions.

Lemma 3.4 *Let Δ be a useful or efficient* metric. *Let $A^{\mathcal{P}}$ be an algorithm querying a distribution ensemble $\mathcal{P}_\theta$ at most q times. Then we have*

$$\Delta(A^{\mathcal{Q}}, \mathcal{R}) \leq \Delta(A^{\mathcal{P}}, \mathcal{R}) + q \cdot \Delta(\mathcal{P}_\theta, \mathcal{Q}_\theta)$$

for any distribution $\mathcal{R}$ and any ensemble $\mathcal{Q}_\theta$.

Proof: By triangle inequality, $\Delta(A^{\mathcal{Q}}, \mathcal{R}) \leq \Delta(A^{\mathcal{P}}, \mathcal{R}) + \Delta(A^{\mathcal{P}}, A^{\mathcal{Q}})$. Let $(\theta_i)_i$ (resp. $(\tilde{\theta}_i)_i$) be the sequence of queries made by $A^{\mathcal{P}}$ (resp. $A^{\mathcal{Q}}$). By data processing inequality $\Delta(A^{\mathcal{P}}, A^{\mathcal{Q}}) \leq \Delta((\theta_i, \mathcal{P}_i)_i, (\tilde{\theta}_i, \mathcal{Q}_i)_i)$. The rest follows from sub-additivity and Lemma 3.2. $\square$

By letting A be the algorithm that performs the convolution as in Theorem 2.1 and applying Lemma 3.4 to it with $\mathcal{P}_i = \mathcal{D}_{\Lambda,\mathbf{c}_i,s_i}$ and approximate distributions $\mathcal{Q}_i = \tilde{\mathcal{D}}_{\Lambda,\mathbf{c}_i,s_i}$, we can show that convolving approximate discrete Gaussians results in good approximations of the expected discrete Gaussian. Furthermore, we can also apply Lemma 3.4 to Theorem 2.2, if we have a bound on the approximation of the second sampler for *any* center c_2.

As an example, consider again the statistical distance Δ_{SD}. By applying Lemma 3.4 to the convolutions in Theorem 2.1 (resp. 2.2), the resulting approximation error satisfies:

$$\Delta_{\mathrm{SD}}(A^{\tilde{\mathcal{D}}_{\Lambda,\mathbf{c}_i,s_i}}, \mathcal{D}_{Y,s}) \lesssim 2\epsilon + \sum_i \Delta_{\mathrm{SD}}(\tilde{\mathcal{D}}_{\Lambda,\mathbf{c}_i,s_i}, \mathcal{D}_{\Lambda,\mathbf{c}_i,s_i}).$$

Conveniently, this works recursively: if we use the obtained approximate samples as input to another convolution, the loss in statistical distance is simply additive in the number of convolutions we apply. This shows that using a metric to analyze approximation errors is relatively straight-forward.

Unfortunately, Δ_{SD} is not cryptographically efficient and thus requires high precision to guarantee security. While $\sqrt{\delta_{\mathrm{KL}}}$ allows to improve on that, it is not a metric and thus Lemma 3.4 does not apply. One can still use $\sqrt{\delta_{\mathrm{KL}}}$ to improve on the efficiency by exploiting the metric properties of Δ_{SD}, i.e. one first decomposes the statistical distance of the approximate distribution as in the previous paragraph, and then bounds the individual parts using property 3. But as we start working with more complex and recursive algorithms, this method becomes more involved. One needs to be careful to not rely on typical metric properties when analyzing algorithms using $\sqrt{\delta_{\mathrm{KL}}}$, like triangle inequality and symmetry. We found it much more convenient to use a cryptographically useful and efficient *metric* Δ. This allows to carry out the analysis using only Δ, and directly claim bit security of $2 \log \Delta(\mathcal{P}_\theta, \mathcal{Q}_\theta)$ by Lemma 3.3.

4 A New Closeness Metric

In this section we introduce a new measure of closeness between probability distributions which combines the ease of use of a metric with the properties of divergences that allow to obtain sharper security bounds. More specifically, we provide an efficient metric with a simple definition.

Definition 2 *The* max-log *distance between two distributions $\mathcal{P}$ and $\mathcal{Q}$ over the same support S is*

$$\Delta_{\mathrm{ML}}(\mathcal{P}, \mathcal{Q}) \quad = \quad \max_{x \in S} \left| \ln \mathcal{P}(x) - \ln \mathcal{Q}(x) \right|.$$

For convenience, we also write $\Delta_{\text{ML}}(p,q) = |\ln p - \ln q|$ for any two positive reals p and q. It is easy to see that Δ_{ML} is a metric.

Lemma 4.1 *Δ_{ML} is a metric, i.e., it is symmetric ($\Delta_{\text{ML}}(\mathcal{P},\mathcal{Q}) = \Delta_{\text{ML}}(\mathcal{Q},\mathcal{P})$), positive definite ($\Delta_{\text{ML}}(\mathcal{P},\mathcal{Q}) \geq 0$ with equality if and only if $\mathcal{P} = \mathcal{Q}$), and it satisfies the triangle inequality ($\Delta_{\text{ML}}(\mathcal{P},\mathcal{Q}) \leq \Delta_{\text{ML}}(\mathcal{P},\mathcal{R}) + \Delta_{\text{ML}}(\mathcal{R},\mathcal{Q})$).*

Proof: All properties are inherited from the infinity norm, simply by noticing that $\Delta_{\text{ML}}(\mathcal{P},\mathcal{Q}) = \|f(\mathcal{P}) - f(\mathcal{Q})\|_\infty$ for some function $f(\mathcal{P}) = (\ln \mathcal{P}(x))_x$. □

We note that in the regime close to 0, Δ_{ML} is essentially equal to δ_{RE}.

Lemma 4.2 *For any two positive real p and q,*

$$\begin{aligned} \Delta_{\text{ML}}(p,q) &\leq -\ln(1-\delta_{\text{RE}}(p,q)) \lesssim \delta_{\text{RE}}(p,q) && (2)\\ \delta_{\text{RE}}(p,q) &\leq \exp(\Delta_{\text{ML}}(p,q)) - 1 \lesssim \Delta_{\text{ML}}(p,q). && (3) \end{aligned}$$

The same bounds hold for $\Delta_{\text{ML}}(\mathcal{P},\mathcal{Q})$ and $\delta_{\text{RE}}(\mathcal{P},\mathcal{Q})$ for any two distributions $\mathcal{P},\mathcal{Q}$ over the same support S.

Proof: Let $\epsilon = \delta_{\text{RE}}(p,q)$, so that $(q/p) \in (1\pm\epsilon)$. It follows that $\Delta_{\text{ML}}(p,q) = |\ln(p/q)| \leq \max[\ln(1+\epsilon), \ln(1/(1-\epsilon))] = -\ln(1-\epsilon)$. This proves (2). For (3), let $\epsilon = \Delta_{\text{ML}}(p,q)$, so that $\max(p/q, q/p) \leq \exp(\epsilon)$. If $p < q$, then $\delta_{\text{RE}}(p,q) = (q/p) - 1 \leq \exp(\epsilon) - 1$. Else, $p \geq q$, and $\delta_{\text{RE}}(p,q) = 1 - (q/p) \leq (p/q) - 1 \leq \exp(\epsilon) - 1$. The same bounds for distributions easily follow by taking the maximum of $\Delta_{\text{ML}}(\mathcal{P}(x),\mathcal{Q}(x))$ and $\delta_{\text{RE}}(\mathcal{P}(x),\mathcal{Q}(x))$ when x ranges over the support of the two distributions. □

The next two lemmas prove that Δ_{ML} is an efficient metric.

Lemma 4.3 *Δ_{ML} satisfies the sub-additivity property (for joint distributions) and data processing inequality.*

Proof: We start by proving the subadditivity property for sequences of length two. The general case follows by induction. By triangle inequality,

$$\begin{aligned} \Delta_{\text{ML}}((X_1,X_2),(Y_1,Y_2)) &\leq \Delta_{\text{ML}}((X_1,X_2),(X_1,[Y_2 \mid Y_1 = X_1]))\\ &+ \Delta_{\text{ML}}((X_1,[Y_2 \mid Y_1 = X_1]),(Y_1,Y_2)) \end{aligned}$$

where $(X_1,[Y_2 \mid Y_1 = X_1])$ is the distribution that selects a pair (x,y) by first choosing x with probability $\Pr\{X_1 = x\}$, and then y with probability $\Pr\{Y_2 = y \mid Y_1 = x\}$. By definition,

$$\begin{aligned} \Delta_{\text{ML}}((X_1,[Y_2 \mid Y_1 = X_1]),(Y_1,Y_2)) &= \max_{(x,y)} \left|\ln \frac{\Pr\{X_1 = x\}\cdot\Pr\{Y_2 = y \mid Y_1 = x\}}{\Pr\{Y_1 = x\}\cdot\Pr\{Y_2 = y \mid Y_1 = x\}}\right|\\ &= \Delta_{\text{ML}}(X_1,Y_1). \end{aligned}$$

and also

$$\begin{aligned} \Delta_{\text{ML}}((X_1,X_2),(X_1,[Y_2 \mid Y_1 = X_1])) &= \max_{(x,y)} \left|\ln \frac{\Pr\{X_1 = x\}\cdot\Pr\{X_2 = y \mid X_1 = x\}}{\Pr\{X_1 = x\}\cdot\Pr\{Y_2 = y \mid Y_1 = x\}}\right|\\ &= \max_x \Delta_{\text{ML}}([X_2 \mid X_1 = x],[Y_2 \mid Y_1 = x]) \end{aligned}$$

This proves subadditivity for joint distributions.

For the data processing inequality, let $\mathcal{P}$ and $\mathcal{Q}$ be two probability distributions with support S and $f\colon S \mapsto T$ be any (deterministic) function. Then

$$\begin{aligned}
\Delta_{\text{ML}}(f(\mathcal{P}), f(\mathcal{Q})) &= \max_{j\in T} |\ln \Pr[f(\mathcal{P}) = j] - \ln Pr[f(\mathcal{Q}) = j]| \\
&= \max_{j\in T} \ln\left(\max\left\{\frac{\sum_{i\in f^{-1}(j)} \mathcal{P}(i)}{\sum_{i\in f^{-1}(j)} \mathcal{Q}(i)}, \frac{\sum_{i\in f^{-1}(j)} \mathcal{Q}(i)}{\sum_{i\in f^{-1}(j)} \mathcal{P}(i)}\right\}\right) \\
&\leq \max_{j\in T} \ln\left(\max\left\{\max_{i\in f^{-1}(j)} \frac{\mathcal{P}(i)}{\mathcal{Q}(i)}, \max_{i\in f^{-1}(j)} \frac{\mathcal{Q}(i)}{\mathcal{P}(i)}\right\}\right) \\
&= \max_{i\in S} \ln\left(\max\left\{\frac{\mathcal{P}(i)}{\mathcal{Q}(i)}, \frac{\mathcal{Q}(i)}{\mathcal{P}(i)}\right\}\right) = \Delta_{\text{ML}}(\mathcal{P}, \mathcal{Q})
\end{aligned}$$

where the inequality holds, because $(\sum_i a_i)/(\sum_i b_i) \leq \max_i(a_i/b_i)$ for all $b_i \geq 0$. The result for randomized functions follows by treating the random coins as explicit input and combining the above with the sub-additivity property. □

Finally, we show that Δ_{ML} also satisfies the pythagorean probability preservation property for any parameter $\lambda \leq \frac{1}{3}$.

Lemma 4.4 *For distributions $\mathcal{P}_i$ and $\mathcal{Q}_i$ over support $\prod_i S_i$, if $\Delta_{\text{ML}}(\mathcal{P}_i \mid a_i, \mathcal{Q}_i \mid a_i) \leq 1/3$ for all i and $a_i \in \prod_{j<i} S_j$, then*

$$\Delta_{\text{SD}}((\mathcal{P}_i)_i, (\mathcal{Q}_i)_i) \leq \|(\max_{a_i} \Delta_{\text{ML}}(\mathcal{P}_i \mid a_i, \mathcal{Q}_i \mid a_i))_i\|_2.$$

Proof: First, we observe that under the condition $\Delta_{\text{ML}}(\mathcal{P}, \mathcal{Q}) \leq 1/3$, we have $\delta_{\text{KL}}(\mathcal{P}, \mathcal{Q}) \leq 2\Delta_{\text{ML}}(\mathcal{P}, \mathcal{Q})^2$. This can be checked using Equation (3) as follows. Let $x = \Delta_{\text{ML}}(\mathcal{P}, \mathcal{Q}) \leq 1/3$. Applying Lemma 2.1 with $\mu = e^x - 1$, we get

$$\delta_{\text{KL}}(\mathcal{P}, \mathcal{Q}) \leq \frac{(e^x - 1)^2}{2(2 - e^x)^2} \leq 2x^2,$$

where the last inequality is implied by $(e^x - 1)(1 + 1/(2x)) \leq 1$, which can be verified using the convexity bound $e^x - 1 \leq (e^{\frac{1}{3}} - 1)3x$ (valid for $x \in [0, 1/3]$) as follows:

$$(e^x - 1) \cdot \left(1 + \frac{1}{2x}\right) \leq (e^{\frac{1}{3}} - 1) \cdot (3x + 1.5) \leq (e^{\frac{1}{3}} - 1) \cdot 2.5 \approx 0.99.$$

Now that we have established the bound $\delta_{\text{KL}}(\mathcal{P}, \mathcal{Q}) \leq 2\Delta_{\text{ML}}(\mathcal{P}, \mathcal{Q})^2$, we can use Pinsker's inequality and the sub-additivity of δ_{KL} (which directly follows from what is often referred to as the *chain rule*) to get

$$\begin{aligned}
\Delta_{\text{SD}}((\mathcal{P}_i)_i, (\mathcal{Q}_i)_i) &\leq \sqrt{\delta_{\text{KL}}((\mathcal{P}_i)_i, (\mathcal{Q}_i)_i)/2} \\
&\leq \sqrt{\frac{1}{2}\sum_i \max_{a_i} \delta_{\text{KL}}(\mathcal{P}_i \mid a_i, \mathcal{Q}_i \mid a_i)} \\
&\leq \sqrt{\sum_i \max_{a_i} \Delta_{\text{ML}}(\mathcal{P}_i \mid a_i, \mathcal{Q}_i \mid a_i)^2} \\
&= \|(\max_{a_i} \Delta_{\text{ML}}(\mathcal{P}_i \mid a_i, \mathcal{Q}_i \mid a_i))_i\|_2.
\end{aligned}$$

□

It follows that we can instantiate Lemma 3.4 with it to analyze the increase of approximation error if applying multiple convolutions to approximate samples. We make this explicit by reformulating Theorem 2.1 and 2.2 in terms of the max-log distance and approximate distributions (following Lemma 3.4), specializing them to our setting.

Corollary 4.1 *Let $\mathbf{z} \in \mathbb{Z}^m$ be a nonzero integer vector with $gcd(\mathbf{z}) = 1$ and $\boldsymbol{s} \in \mathbb{R}^m$ with $s_i \geq \sqrt{2}\|\mathbf{z}\|_\infty \eta_\epsilon(\mathbb{Z})$ for all $i \leq m$. Let y_i be independent samples from $\tilde{\mathcal{D}}_{\mathbb{Z},s_i}$, respectively, with $\Delta_{\text{ML}}(\mathcal{D}_{\mathbb{Z},s_i}, \tilde{\mathcal{D}}_{\mathbb{Z},s_i}) \leq \mu_i$ for all i. Let $\tilde{\mathcal{D}}_{\mathbb{Z},s}$ be the distribution of $y = \sum z_i y_i$. Then $\Delta_{\text{ML}}(\mathcal{D}_{\mathbb{Z},s}, \tilde{\mathcal{D}}_{\mathbb{Z},s}) \lesssim 2\epsilon + \sum_i \mu_i$.*

Corollary 4.2 *Let $s_1, s_2 > 0$, with $s^2 = s_1^2 + s_2^2$ and $s_3^{-2} = s_1^{-2} + s_2^{-2}$. Let $\Lambda = K\mathbb{Z}$ be a copy of the integer lattice $\mathbb{Z}$ scaled by a constant K. For any c_1 and $c_2 \in \mathbb{R}$, denote the distribution of $x_1 \leftarrow x_2 + \tilde{\mathcal{D}}_{c_1 - x_2 + \mathbb{Z}, s_1}$, where $x_2 \leftarrow \tilde{\mathcal{D}}_{c_2+\Lambda,s_2}$, by $\tilde{\mathcal{D}}_{c_1+\mathbb{Z},s}$. If $s_1 \geq \eta_\epsilon(\mathbb{Z})$, $s_3 \geq \eta_\epsilon(\Lambda) = K\eta_\epsilon(\mathbb{Z})$, $\Delta_{\text{ML}}(\mathcal{D}_{c_2+\Lambda,s_2}, \tilde{\mathcal{D}}_{c_2+\Lambda,s_2}) \leq \mu_2$ and $\Delta_{\text{ML}}(\mathcal{D}_{c+\mathbb{Z},s_1}, \tilde{\mathcal{D}}_{c+\mathbb{Z},s_1}) \leq \mu_2$ for any $c \in \mathbb{R}$, then $\Delta_{\text{ML}}(\mathcal{D}_{c_1+\mathbb{Z},s}, \tilde{\mathcal{D}}_{c_1+\mathbb{Z},s}) \lesssim 4\epsilon + \mu_1 + \mu_2$.*

Relationship to Other Measures. The max-log distance is closely related to the Rényi divergence of order ∞ and shares many of its properties, including a multiplicative probability preservation: $Pr_{X \leftarrow \mathcal{P}}[E] \geq Pr_{X \leftarrow \mathcal{Q}}[E] / \exp(\Delta_{\text{ML}}(\mathcal{P}, \mathcal{Q}))$ [5]. This can be useful for other definitions of bit security than Definition 1. In particular, consider a setting where the number of queries to the distribution is bounded by some fixed number q. This seems reasonable in many applications since queries often require interaction with an honest user. In this setting, it is easy to show that the success probability of an adversary can only increase by a multiplicative factor of $\exp(q\Delta_{\text{ML}}(\mathcal{P}_\theta, \mathcal{Q}_\theta))$ using sub-additivity, data processing inequality, and multiplicative probability preservation. This shows that as long as $\Delta_{\text{ML}}(\mathcal{P}, \mathcal{Q}) < 1/q$, one loses almost no security. In a subsequent work [40], Prest shows that by generalizing Lemma 2.1 to Rényi divergences of arbitrary order, one can achieve tighter bounds ($\Delta_{\text{ML}}(\mathcal{P}, \mathcal{Q}) \approx \sqrt{1/q}$) in this setting by optimizing over the order of the divergence. In cases where the number of queries q is only bounded by the running time of the adversary, as in Definition 1, the bound of [40] reduces to the one in Lemma 3.3.

It has also been noted that the Rényi divergence is related to the notion of differential privacy. More specifically, an algorithm $A(D)$, taking a database D as input, is ϵ-differentially private if the Rényi divergence of order ∞ between the output distributions of $A(D_1)$ and $A(D_2)$ is less than ϵ for any two neighboring databases D_1 and D_2. Since *neighborhood* is often defined using a symmetric relation on the set of databases, this is equivalent to a formulation using the max-log distance. Finally, the techniques used in [40] are related to *advanced composition theorems* in the differential privacy terminology. For more details we refer the reader to [32] and references therein.

5 Sampling the Integers

In this section we describe and analyze our new algorithm to sample the discrete Gaussian distribution. The entire algorithm SampleZ is presented in Algorithm 1. In Sect. 5.1 and 5.2, we analyze the subroutines SampleI and SampleC, which may already be directly useful in some applications. Then, in Sect. 5.3, we analyze the full algorithm SampleZ. All algorithms assume access to a base sampler SampleB to approximate the distribution $\mathcal{D}_{c_i+\mathbb{Z},s_0}$, for a small and fixed set of values for the coset c_i and one fixed s_0. Any algorithm can be used as a base sampler, provided it produces distributions $\tilde{\mathcal{D}}_{c_i+\mathbb{Z},s_0}$ within a small distance $\Delta_{\text{ML}}(\tilde{\mathcal{D}}_{c_i+\mathbb{Z},s_0}, \mathcal{D}_{c_i+\mathbb{Z},s_0}) \leq \mu$ from the exact Gaussian $\mathcal{D}_{c_i+\mathbb{Z},s_0}$. By Lemma 4.2, this is essentially equivalent to approximating the Gaussian probabilities with a relative error bound of μ. The reader is referred to Sect. 6.2 for a possible choice of SampleB.

$\textsc{SampleZ}_{b,k,\max}(c, s)$
  $x \leftarrow \textsc{SampleI}(\max)$
  $K \leftarrow \sqrt{s^2 - \bar{s}^2}/s_{\max}$
  $c' \leftarrow \lfloor c + Kx \rceil_k$
  $y \leftarrow \textsc{SampleC}_{b,s_0}(c')$
  return y

$\textsc{SampleC}_b(c \in b^{-k}\mathbb{Z})$
  if $k = 0$
    return 0
  $g \leftarrow b^{-k+1} \cdot \textsc{SampleB}_{s_0}(b^{k-1}c)$
  return $g + \textsc{SampleC}_b(c - g \in b^{-k+1}\mathbb{Z})$

$\textsc{SampleI}(i)$
  if $i = 0$
    $x \leftarrow \textsc{SampleB}_{s_0}(0)$
    return x
  $x_1 \leftarrow \textsc{SampleI}(i-1)$
  $x_2 \leftarrow \textsc{SampleI}(i-1)$
  $y = z_i x_1 + \max(1, z_i - 1)x_2$
  return y

Algorithm 1: A sampling algorithm for $\mathcal{D}_{c+\mathbb{Z},s}$ for arbitrary c and s. Definitions for z_i and s_i as in (4) and (5) and $\bar{s}$ as in (6). SampleB is an arbitrary base sampler for $\mathcal{D}_{c+\mathbb{Z},s_0}$ with fixed s_0 and small number of cosets $c + \mathbb{Z}$, where $c \in \mathbb{Z}/b$.

5.1 Large deviations

In this section we show how to efficiently sample $\mathcal{D}_{\mathbb{Z},s}$ for an arbitrarily large $s \gg \eta_\epsilon(\mathbb{Z})$ using samples from $\mathcal{D}_{\mathbb{Z},s_0}$ for some small fixed value of $s_0 \geq \sqrt{2}\eta_\epsilon(\mathbb{Z})$. For this we make use of convolution to combine the samples from the basic sampler to yield a distribution with larger noise parameter. The algorithm accomplishing this is given in Algorithm 1 as SampleI.

Lemma 5.1 *For a given value of $s_0 \geq 4\sqrt{2}\eta_\epsilon(\mathbb{Z})$ define the following sequence of values[2] for $i > 0$:*

$$z_i = \left\lfloor \frac{s_{i-1}}{\sqrt{2}\eta} \right\rceil \tag{4}$$

$$s_i^2 = (z_i^2 + \max((z_i - 1)^2, 1))s_{i-1}^2 \tag{5}$$

If $\Delta_{\text{ML}}(\mathcal{D}_{\mathbb{Z},s_0}, \textsc{SampleB}_{s_0}(0)) \leq \mu$, then $\Delta_{\text{ML}}(\mathcal{D}_{\mathbb{Z},s_i}, \textsc{SampleI}(i)) \leq (\mu + 2\epsilon)2^i$ and the running time of SampleI *is at most 2^i plus 2^i invocations of* SampleB*. Finally, $s_i(s_0) \geq 2^{2^i}$, implying $i \leq \lceil \log\log s \rceil$ is sufficient to achieve a given target s.*

*Proof:*Note that SampleI repeatedly invokes Corollary 4.1. The conditions of Corollary 4.1 are met by definition of z_i (Equation (4)), so every application incurs a loss in Δ_{ML} of 2ϵ by Corollary 4.1. The bound on the number of base samples and convolutions is immediate.

We conclude by proving the statement $s_i(s_0) \geq 2^{2^i}$ under the conditions of the lemma. Let $\eta = \eta_\epsilon(\mathbb{Z})$. By definition we have $z_i \geq \frac{s_{i-1}}{\sqrt{2}\eta} - 1$ and so

$$s_i^2 \geq 2s_{i-1}^2 \left(\frac{s_{i-1}}{\sqrt{2}\eta} - 2\right)^2$$

and so

$$s_i \geq \sqrt{2}s_{i-1}\left(\frac{s_{i-1}}{\sqrt{2}\eta} - 2\right) \geq s_{i-1}^2\left(\frac{1}{\eta} - \frac{2\sqrt{2}}{s_{i-1}}\right) \geq s_{i-1}^2\left(\frac{1}{\eta} - \frac{2\sqrt{2}}{s_0}\right) \geq \frac{s_{i-1}^2}{2\eta}.$$

[2]Notice that the values in (4) and (5) depend both on the index i and the initial s_0, so we will write them as $z_i(s_0)$ and $s_i(s_0)$ when we need to emphasize this dependency.

Equivalently,

$$\log s_i \geq 2 \log s_{i-1} - \log 2\eta.$$

Unrolling the recursion, we obtain

$$\log s_i \geq 2^i \log s_0 - \left(\sum_{j=0}^{i-1} 2^j \right) \log 2\eta = 2^i \log s_0 - (2^i - 1) \log 2\eta \geq 2^i (\log s_0 - \log 2\eta) \geq 2^i$$

and so $s_i \geq 2^{2^i}$. $\square$

The algorithm SampleI will overshoot the noise parameter, but in many applications (including ours further below) this is enough. In fact, for us it will not matter by how much we overshoot a given target s, as we will show in the following sections how to adjust the noise parameter to obtain a sample from a specific target distribution (with arbitrary center).

SampleCenteredGaussian(s)
  Select largest i such that $s_i < s$
  $x_1 \leftarrow$ SampleI(i)
  $x_2 \leftarrow$ SampleI(i)
  $z \leftarrow \left\lceil \frac{1}{2} \left(1 + \sqrt{2 \left(\frac{s}{s_i} \right)^2 - 1} \right) \right\rceil$
  return $zx_1 + (z-1)x_2$

Algorithm 2: A sampling algorithm for $\mathcal{D}_{\mathbb{Z},\tilde{s}}$ for some $\tilde{s}$ not much larger than s. Definition of s_i as in (5).

If all we are interested in is the centered Gaussian distribution with a specific noise parameter not much larger than a certain target width, as is the case in many applications, it is relatively easy to adapt the algorithm to get closer to the target s. One way of doing this is to adjust z_i in the top level of the recursion to yield something closer to s. This is demonstrated by Algorithm 2, for which the following corollary establishes a bound on the size of the resulting noise parameter.

Corollary 5.1 *If* $\Delta_{\text{ML}}(\text{SampleI}(i), \mathcal{D}_{\mathbb{Z},s_i}) \lesssim \mu$ *for the largest* i *such that* $s_i \leq s$ *and* $s \geq s_0 \geq \sqrt{2}\eta_\epsilon(\mathbb{Z})$*, then* $\Delta_{\text{ML}}(\text{SampleCenteredGaussian}(s), \mathcal{D}_{\mathbb{Z},\tilde{s}}) \lesssim 2\mu + 2\epsilon$ *for some* $\tilde{s}$ *such that* $s \leq \tilde{s} \leq \sqrt{5}s$.

Proof: First note that $s_i < s$ implies $z \geq 2$. The choice of z and s_i now guarantees that Corollary 4.1 is applicable and that $(z-1)^2 + (z-2)^2 < \frac{s^2}{s_i^2} \leq z^2 + (z-1)^2$. Since $\tilde{s}^2 = (z^2 + (z-1)^2) s_i^2$ this establishes the lower bound and shows that $\tilde{s}^2 \leq \frac{z^2+(z-1)^2}{(z-1)^2+(z-2)^2} s^2$. The upper bound follows from the fact that the ratio $\frac{z^2+(z-1)^2}{(z-1)^2+(z-2)^2}$ is decreasing in z and equals 5 for $z = 2$.

The bound on the Δ_{ML} distance is immediate from Corollary 4.1. $\square$

Note that the constant $\sqrt{5}$ in Corollary 5.1 follows from the worst case where $z = 2$. Using a little more care in the choice of small coefficients, the bound can be improved to $\sqrt{2}$, but for a simpler exposition we omitted this optimization. However, it will not be possible to get arbitrarily close to any target s if given a fixed s_0, but if the target s is fixed we can always choose a suitable small s_0 such that the target distribution will be generated exactly.

For a fixed s_0, $z_i(s_0)$ and $s_i(s_0)$ are fixed, so one can precompute s_i and corresponding z_i for a small set of i. As Lemma 5.1 shows, the s_i grow very rapidly so only a very small number ($\sim \log \log s$) of precomputed values are necessary to generate extremely wide distributions. If the target s is fixed, only the coefficients z_i need to be stored.

5.2 Arbitrary center

We now show how to sample from an arbitrary coset $c+\mathbb{Z}$ using samplers for only a small number of cosets. We assume c is given as a k digit number in base b between 0 and 1. The parameter k dictates the trade-off between running time and output precision, while the basis b determines the number of cosets the base sampler SampleB needs to be able to sample from.

The idea of our new algorithm SampleC (see Algorithm 1) is to round the center randomly digit by digit to finally obtain a sample from $c+\mathbb{Z}$. Every rounding operation consumes a sample from one of b cosets of $\mathbb{Z}$ (where b is a parameter). To show correctness, we iteratively use a convolution theorem.

While this process of iterative rounding increases the noise of the output distribution, this increase is minor as the following lemma shows.

Lemma 5.2 *Let $2 \leq b \in \mathbb{Z}$ be a base, $s_0 \geq (\sqrt{(b+1)/b})\eta_\epsilon(\mathbb{Z})$ and $c \in b^{-k}\mathbb{Z}$. If*

$$\Delta_{\mathrm{ML}}(\mathcal{D}_{c_i+\mathbb{Z},s_0}, \textsc{SampleB}_{s_0}(c_i)) \leq \mu$$

for all $c_i \in \mathbb{Z}/b$, then $\Delta_{\mathrm{ML}}(\textsc{SampleC}_b(c), \mathcal{D}_{c+\mathbb{Z},\bar{s}}) \lesssim (4\epsilon+\mu)k$ where

$$\bar{s} = s_0 \left(\sqrt{\sum_{i=0}^{k-1} b^{-2i}} \right). \tag{6}$$

Proof: The proof follows by induction and Corollary 4.2. For $k = 1$ the claim is obviously true. For $k > 1$, invoke the induction hypothesis and apply Corollary 4.2 with $s_1 = s_0\sqrt{\sum_{i=0}^{k-2} b^{-2i}}$, $s_2 = s_0/b^{k-1}$, $\Lambda = b^{-k+1}\mathbb{Z}$, $c_2 = b^{-k}[c]_k$ (where $[c]_k$ is the k-th digit in the b-ary expansion of c), and $c_1 = c$.

It remains to show that the conditions on the noise parameters are met. First note that $\sum_{i=0}^{k} b^{-2i} \geq 1$ for all $k \geq 1$, and so $s_1 \geq s_0 > \eta_\epsilon(\mathbb{Z})$.

Then we have

$$\begin{aligned} s_3^{-2} = s_1^{-2} + s_2^{-2} &= s_0^{-2}\left(\left(\sum_{i=0}^{k-2} b^{-2i}\right)^{-1} + b^{2(k-1)}\right) \\ &= s_0^{-2}\left(\frac{1-b^{-2}}{1-b^{-2(k-1)}} + b^{2(k-1)}\right) = s_0^{-2}\frac{b^{2(k-1)}-b^{-2}}{1-b^{-2(k-1)}} \end{aligned}$$

and so

$$s_3 = \sqrt{\frac{1-b^{-2(k-1)}}{b^{2(k-1)}-b^{-2}}}s_0 = \frac{1}{b^{k-1}}\sqrt{\frac{1-b^{-2(k-1)}}{1-b^{-2k}}}s_0 = \frac{1}{b^{k-1}}\sqrt{\frac{b^{2k}-b^2}{b^{2k}-1}}s_0$$

Note that

$$\frac{b+1}{b}\cdot\frac{b^{2k}-b^2}{b^{2k}-1} \geq 1$$

for all $k > 1$, which shows that $s_3 \geq b^{-k+1}\eta_\epsilon(\mathbb{Z}) = \eta_\epsilon(\Lambda)$. □

The parameter b in SampleC offers a trade-off between running time and number of required samplers for cosets of $\mathbb{Z}$. As most efficient samplers require storage for each coset, this is effectively a time-memory trade-off. The larger the base b, the more bits we can round at a time, but that requires more cosets. Note that the running time decreases by a logarithmic factor in b, while the storage requirement increases linearly with b.

5.2.1 Reducing the number of required samples

Recall from the previous section that the parameter k determines the trade-off between running time and output precision: the larger k, the closer the approximation of the centers and thus the better the output distribution, but the number of required base samples and the running time grow linearly with k. We now show that by using a biased coin flip we can speed up the algorithm by a factor 2 while maintaining a good approximation.

Lemma 5.3 *Let $s \geq \eta_\epsilon(\mathbb{Z})$ and $b, k \in \mathbb{Z}$ such that $\tau = b^{-k} \leq (4\pi)^{-1}$. Then*

$$\Delta_{\mathrm{ML}}(\mathcal{D}_{\mathbb{Z},c,s}, \mathcal{D}_{\mathbb{Z},\lfloor c \rceil_k,s}) \lesssim \pi^2\tau^2 + 2\epsilon = \pi^2/b^{2k} + 2\epsilon,$$

where $\mathcal{D}_{\mathbb{Z},\lfloor c \rceil_k,s}$ is the distribution of the process of computing $c' = \lfloor c \rceil_k$ and then returning a sample from $\mathcal{D}_{\mathbb{Z},c',s}$.

To prove the lemma, we first observe that linear functions can approximate the Gaussian function well on small enough intervals.

Lemma 5.4 *For any x_1, x_2 with $x_2 - x_1 = \tau$, $|x_1|, |x_2| \leq ts$ for some $t \geq 1$ and $x \in [x_1, x_2]$, we have*

$$\delta_{\mathrm{RE}}\left(\rho_s(x), \frac{x - x_1}{\tau}\rho_s(x_2) + \frac{x_2 - x}{\tau}\rho_s(x_1)\right) \leq \frac{\pi^2 t^2 \tau^2}{2s^2} e^{\frac{2\pi\tau t}{s}}.$$

In particular, if $\tau \leq \frac{s}{4\pi t}$, the bound on the right hand side is less than $\frac{\pi^2 t^2 \tau^2}{s^2}$.

Proof: By linear interpolation,

$$\left|\rho_s(x) - \left(\frac{x - x_1}{\tau}\rho_s(x_2) + \frac{x_2 - x}{\tau}\rho_s(x_1)\right)\right| \leq \frac{\tau^2}{8} \max_{x_1 \leq x' \leq x_2} |\rho_s''(x')|$$

Observe that

$$\rho_s''(x) = \left(\frac{2\pi x^2}{s^2} - 1\right)\frac{2\pi}{s^2}\rho_s(x)$$

implying that $\|\rho_s''(x')\| \leq \max(\frac{2\pi x'^2}{s^2}, 1)\frac{2\pi}{s^2}\rho_s(x') \leq \frac{4\pi^2 t^2}{s^2}\rho(x')$. Finally note that if $x'^2 \geq x^2$, then $\rho_s(x') \leq \rho_s(x)$. Otherwise,

$$\frac{\rho_s(x')}{\rho_s(x)} = e^{-\pi(\frac{x'^2 - x^2}{s^2})} = e^{\pi(\frac{x^2 - x'^2}{s^2})} = e^{\pi(\frac{(x - x')(x + x')}{s^2})} \leq e^{\frac{2\pi\tau t}{s}}$$

concluding the proof. □

Proof:[of Lemma 5.3] We set $t = \eta_\epsilon(\mathbb{Z})$, which allows us to treat $\mathcal{D}_{c+\mathbb{Z},s}$ as a ts-bounded distribution. If we assume that $s \geq \eta_\epsilon(\mathbb{Z})$ for some negligible ϵ, we can conclude that Lemma 5.4 also holds for the respective

distributions, since $\rho_s(c+\mathbb{Z}) \approx s$ for any c, i.e. with $c_1 = \lfloor c \rfloor_k$ and $c_2 = \lceil c \rceil_k$:

$$\begin{aligned}
\Delta_{\text{ML}}(\mathcal{D}_{\mathbb{Z},c,s}, \mathcal{D}_{\mathbb{Z},\lfloor c \rceil_r,s}) &= \max_x \left| \ln \frac{\mathcal{D}_{\mathbb{Z},c,s}(x)}{\mathcal{D}_{\mathbb{Z},\lfloor c \rceil_r,s}(x)} \right| \\
&= \max_x \left| \ln \frac{\mathcal{D}_{\mathbb{Z},c,s}(x)}{\left(\frac{c_2-c}{\tau}\mathcal{D}_{\mathbb{Z},c_1,s}(x) + \frac{c-c_1}{\tau}\mathcal{D}_{\mathbb{Z},c_2,s}(x)\right)} \right| \\
&\leq \max_x \left| \ln \frac{\rho_s(x-c)(1\pm\epsilon)s}{(1\pm\epsilon)s\left(\frac{c_2-c}{\tau}\rho_s(x-c_1) + \frac{c-c_1}{\tau}\rho_s(x-c_2)\right)} \right| \\
&\leq \max_x \left| \Delta_{\text{ML}}\left(\rho_s(x-c), \frac{c_2-c}{\tau}\rho_s(x-c_1) + \frac{c-c_1}{\tau}\rho_s(x-c_2)\right) + \ln\frac{1\pm\epsilon}{1\pm\epsilon} \right| \\
&\lesssim \max_x \delta_{\text{RE}}\left(\rho_s(x-c), \frac{c_2-c}{\tau}\rho_s(x-c_1) + \frac{c-c_1}{\tau}\rho_s(x-c_2)\right) + 2\epsilon \\
&\leq \frac{\pi^2 t^2 \tau^2}{s^2} + 2\epsilon \\
&\lesssim \frac{\pi^2}{b^{2k}} + 2\epsilon
\end{aligned}$$

where we used Lemma 5.4 and Lemma 4.2. □

In combination with SampleC (cf. Algorithm 1), Lemma 5.3 suggests an efficient algorithm to sample from $\mathcal{D}_{\mathbb{Z},c,\bar{s}}$ for fixed s and arbitrary c:

1. write c in base b (which is a parameter of the algorithm) and divide this representation into the $k = \log_b \frac{1}{\tau}$ higher order digits (representing c_{head}) and the rest c_{tail}
2. use c_{tail} to define the bias of a Bernoulli distribution to round c_{head} either up or down
3. return $\text{SampleC}_{b,s_0}(c_{\text{head}} \in b^{-k}\mathbb{Z})$.

These steps correspond to the computation of c' and the following invocation of SampleC in the algorithm SampleZ. The efficiency gain stems from the fact that sampling from a biased Bernoulli distribution is much cheaper than drawing samples from the discrete Gaussian. This allows us to support centers c with arbitrary precision above k with essentially no efficiency loss, since the lower order bits only define the bias of the Bernoulli distribution, which is cheap to implement.

5.3 The Full Sampler

So far we have shown how to generate samples efficiently from $\mathcal{D}_{\mathbb{Z},s_i}$ for potentially very large s_i and how to sample from $\mathcal{D}_{\mathbb{Z},c,\bar{s}}$ for arbitrary $c \in \mathbb{R}$ and a specific $\bar{s}$, both using only b samplers for $\mathcal{D}_{\mathbb{Z},c_i,s_0}$ for $c_i \in b^{-1}\mathbb{Z}$ and fixed $s_0 \geq \eta_\epsilon(\mathbb{Z})$. We now prove correctness of the full sampler, SampleZ, which puts all the pieces together by leveraging Corollary 4.2 yet again.

Lemma 5.5 *Let $b, k \in \mathbb{Z}$ be a base and a precision parameter such that $k > \log_b 4\pi$. If*

- $\Delta_{\text{ML}}(\mathcal{D}_{\mathbb{Z},s_{\max}}, \text{SampleI}(\max)) \leq \mu_i$ *and*
- $\Delta_{\text{ML}}(\mathcal{D}_{c'+\mathbb{Z},\bar{s}}, \text{SampleC}_b(c')) \leq \mu_c$ *for any $c' \in \mathbb{Z}/b^k$ and some $\bar{s} \geq \eta_\epsilon(\mathbb{Z})$,*

then

$$\Delta_{\text{ML}}(\mathcal{D}_{c+\mathbb{Z},s}, \text{SampleZ}_{b,k,\max}(c,s)) \lesssim 6\epsilon + \pi^2/b^{2k} + \mu_i + \mu_c$$

for any c and s such that $1 < s/\bar{s} \leq s_{\max}/\eta_\epsilon(\mathbb{Z})$.

Proof: By Lemma 5.3 and 3.4, $\Delta_{\text{ML}}(\mathcal{D}_{c+Kx}, \text{SAMPLEC}(\lfloor c+Kx \rceil_k)) \leq \pi^2/b^{2k} + 2\epsilon + \mu_c$. By correctness of SAMPLEI (Lemma 5.1), $\Delta_{\text{ML}}(\mathcal{D}_{K\mathbb{Z},Ks_{\max}}, Kx) \leq \mu_i$ (where $x \leftarrow \text{SAMPLEI}(\max)$) and by definition of K we have $s = \sqrt{(Ks_{\max})^2 + \bar{s}^2}$. Now rewrite $\mathcal{D}_{\mathbb{Z},c+Kx,\bar{s}} = c + Kx + \mathcal{D}_{-Kx-c+\mathbb{Z},\bar{s}}$ and apply Corollary 4.2 with $c_2 = 0$, $c_1 = c$, $x_1 = Kx$ and $x_2 = y$ to see that $\Delta_{\text{ML}}(\mathcal{D}_{c+\mathbb{Z},s}, \text{SAMPLEZ}_{b,k,\max}(c,s)) \lesssim 6\epsilon + \pi^2/b^{2k} + \mu_i + \mu_c$, if the conditions in the theorem are met. This can easily be seen to be true from the assumptions on s by the following calculation.

$$\begin{aligned} s_3 = \left((Ks_{\max})^{-2} + \bar{s}^{-2}\right)^{-\frac{1}{2}} &= \left(\frac{1}{s^2 - \bar{s}^2} + \frac{1}{\bar{s}^2}\right)^{-\frac{1}{2}} = \left(\frac{\bar{s}^2(s^2 - \bar{s}^2)}{s^2}\right)^{\frac{1}{2}} \\ &= \frac{\bar{s}}{s}\sqrt{s^2 - \bar{s}^2} \geq \sqrt{s^2 - \bar{s}^2}\eta_\epsilon(\mathbb{Z})/s_{\max} = \eta_\epsilon(K\mathbb{Z}) \end{aligned}$$

□

The running time of SAMPLEZ is obvious: one invocation of SAMPLEI and one of SAMPLEC, which we analyzed in Sect. 5.1 and 5.2, resp., and a few additional arithmetic operations to calculate K and c'. It is worth noting that the computation of K, the most complex arithmetic computation of the entire algorithm, depends only on s. In many applications, for example trapdoor sampling, s is restricted to a relatively small set, which depends on the key. This means that K_s can be precomputed for the set of possible s's allowing to avoid the FP computation at very low memory cost. Finally, the algorithm may approximate the scaling factor K by a value $\tilde{K}$ such that $\delta_{\text{RE}}(\tilde{K}, K) \leq \mu_K$, which results in an approximation of the distribution of width $\tilde{s} = \sqrt{(\tilde{K}s_i)^2 + \bar{s}}$ instead of s. Elementary calculations show that $\Delta_{\text{ML}}(\mathcal{D}_{\mathbb{Z},c,s}, \mathcal{D}_{\mathbb{Z},c,\tilde{s}}) \lesssim 4\pi t^2 \mu_K$ which by triangle inequality adds to the approximation error.

As an example, assume we have an application, where we know that $\bar{s} \leq s \leq 2^{20} = s_{\max}$. It can be checked, that for any base b and $s_0 \geq 4\sqrt{2}\eta_\epsilon(\mathbb{Z})$, the following parameter settings for our algorithm result in

$$\Delta_{\text{ML}}(\mathcal{D}_{\mathbb{Z},c,s}, \text{SAMPLEZ}_{b,k,\max}(c,s)) \leq 2^{-52},$$

and thus in ≥ 100 bits of security by Lemma 3.3:

- $t = \eta_\epsilon(\mathbb{Z}) = 6$, which results in $\epsilon \leq 2^{-112}$
- $\mu = 2^{-60}$, the precision of the base sampler, resulting in $\mu_i \leq 2^{-55}$
- $k = \lceil 30/\log b \rceil$, which results in $\mu_c \leq 2^{-55}$ and $\pi^2/b^{2k} \leq 2^{-56}$
- $\mu_K = 2^{-64}$, the precision of calculating K, resulting in $4\pi t^2 \mu_K \leq 2^{-55}$.

5.4 Online-Offline Phase and Constant-Time Implementation

Note that a large part of the computation time during our convolution algorithm is spent in the base sampler, which is independent of the center and the noise parameter. This allows us to split the algorithm into an offline and an online phase, similar in spirit to Peikert's sampler [37], which gives rise to a number of platform dependent optimizations. The obvious approach is to simply precompute a number of samples for each of the b cosets and combine them in the online phase until we run out. Note that the trade-off now is not only a time-memory trade-off anymore, it is a time-memory-lifetime trade-off for the device that depends on b. Increasing b speeds up the algorithm, but requires to precompute and store samples for more cosets. While it also means that we effectively decrease the number of samples required per output sample, the latter dependence is only logarithmic, while the former is linear in b.

There are a number of other ways to exploit this structure without limiting the lifetime of the device. Most devices that execute cryptographic primitives have idle times (e.g. web servers) which can be used to restock the number of precomputed samples. As another example, one can separate the offline phase (basic sampler) and the online phase (combination phase) into two parallel devices with a shared buffer. While the basic sampler keeps filling the buffer with samples, the online phase can combine these samples into the

desired distribution. An obvious architecture for such a high performance system would implement the base sampler in a highly parallel fashion (e.g. FPGA or GPU) and the online phase on a regular CPU. This shows that in many scenarios the offline phase can be for free.

The separation of offline and online phase also allows for a straight-forward constant-time implementation with very little overhead. A general problem with sampling algorithms in this context is that the running time of the sampler can leak information about the output sample or the input, which clearly hurts security. For a fixed Gaussian, a simple mitigation strategy is to generate the samples in large batches. This approach breaks down in general when the parameters of the target distribution vary per sample and are not known in advance. In contrast, this idea can be used to implement our algorithm in constant time by generating the basic samples in batches in constant time. Note that every output sample requires the exact same number of base samples and convolutions, so the online phase lends itself naturally to a constant-time implementation.

Assume every invocation of SampleZ requires q base samples and let $\hat{t}_0$ be the maximum over $c_i \in \mathbb{Z}/b$ of the expected running time (over the random coins) of the base sampler (computed either by analysis or experimentation). Consider the following algorithm.

Initialization:

- Use the base sampler to fill b buffers of size q, where the i-th buffer stores discrete Gaussian samples $\mathcal{D}_{c_i+\mathbb{Z},s_o}$ for all $c_i \in \mathbb{Z}/b$.

Query phase:

- On input c and s, call $\text{SampleZ}(c,s)$, where $\text{SampleB}_{s_0}(c_i)$ simply reads from the respective buffer.
- Call the base sampler q times to restock the buffers and pad the running time of this step to $T = q\hat{t}_0 + O(\sqrt{\kappa q})$.

Note that the restocking of base samples in the query phase runs in constant time with overwhelming probability, which follows from Hoeffding's inequality (the constant in the O-notation depends on the worst-case running time of the base sampler). It follows, that the query phase runs in constant time if all the arithmetic operations in SampleZ are implemented in constant time and the randomized rounding operation is converted to constant time, both of which are easy to achieve.

The amortized overhead is only $O(\sqrt{\kappa/q})$, where q is the number of base samples required per output sample. This can be further reduced, if enough memory for larger buffers is available. Finally, the separation of online and offline phase into different independent systems or precomputation of the offline phase allow for an even more convenient constant-time implementation: One only needs to convert the arithmetic operations and the coin flip into constant time. This incurs only a minimal penalty in running time.

6 Applications and Comparison

We first give a short overview of existing sampling algorithms (Sect. 6.1) and select a suitable one as our base sampler, before we describe the experimental study.

6.1 Brief Survey of Existing Samplers

All of the currently known samplers can be categorized into two types[3]: rejection-based samplers and tree traversal algorithms. Table 1 summarizes the existing sampling algorithms and their properties in comparison to our work. The table does not contain a column with the running time, since this depends on a lot of factors (speed of FP arithmetic vs memory access vs randomness etc.), but for the rejection-based samplers, the rejection rate can be thought of as a measure of the running time. Tree-traversal algorithms should be

[3]Technically, even rejection-based samplers can be thought of as tree traversal algorithms, but this is not as natural for them, hence our categorization.

Table 1: Comparison of Sampling Algorithms, starting with rejection-based sampler, followed by tree-traversal samplers and finally Algorithm 1. The column exp(·) indicates if the algorithm requires to evaluate exp(·) online. The column "Generic" refers to the property of being able to produce samples from discrete Gaussians with different parameters not known before precomputation (i.e. which may vary from query to query). The security parameter is denoted by κ.

Algorithm	Memory	Rejection Rate	exp(·)	Generic
Rejection Sampling [22]	0	$\sim .9$	Yes	Yes
Discrete Ziggurat [11]	var	var	Yes	No
Bernoulli-type [14]	$O(\kappa \log s)$	$\sim .5$	No	No
Karney [25]	0	$\sim .5$	No	Yes
Knuth-Yao [18]	$O(\kappa s)$	-	No	No
Inversion Sampling [37]	$O(\kappa s)$	-	No	No
Our work	var	-	No	Yes

thought of as much faster than rejection based samplers. A more concrete comparison on a specific platform will be given in Sect. 6.4 and Sect. 6.6.

6.2 The Base Sampler

We first consider the problem of generating samples from $\mathcal{D}_{\mathbb{Z},c,s}$ when $s = O(\eta_\epsilon(\mathbb{Z}))$ is relatively small and c is fixed. We are interested in the amortized cost of sample generation, where we want to generate a large batch of samples.

We first observe that we are sampling from a relatively narrow Gaussian distribution, so memory will not be a concern for us. For example, assume we choose $s \approx 34 > 4\sqrt{2}\eta_\epsilon(\mathbb{Z})$ for reasonable ϵ, and the tailbound parameter $t = 6$ and store all probabilities, i.e. $\mathcal{D}_{c+\mathbb{Z},s}(i)$ for all $0 \leq i \leq ts$ with $i \in c + \mathbb{Z}$, with 64 bit precision. Then we obtain a memory requirement of only ~ 1.5kb for each of the b cosets. Note that storing half the probability table is sufficient in this case, which is obvious if $c \in \{0, 1/2\}$, but is also true for other c since we can exploit symmetries in the different tables that we store. If indeed less memory is available, one can reduce $s \geq \sqrt{2}\eta(\mathbb{Z})$, which is the minimum to be usable for our algorithms. This will come at a moderate cost in performance.

Finally, the algorithm can also be implemented using a sampler for only the 0-coset and noise parameter bs_0 by bucketing the samples from different cosets in intermediate buffers. This approach has the advantage of being potentially simpler, but can make constant time implementations more troublesome (see Section 5.4).

Since we want to generate a large number of samples, our main criteria for the suitability of an algorithm is its expected running time. For any algorithm, this is lower bounded by the entropy of $\mathcal{D}_{\mathbb{Z},c,s}$, so a natural choice is (lazy) inversion sampling [37] or Knuth-Yao [18], since both are (close to) randomness optimal and their running time is essentially the number of random bits they consume, hence providing us with an optimal algorithm for our purpose. In fact, Knuth-Yao is a little faster than inversion sampling, so we focus on that.

6.3 Setup of Experimental Study

There are a number of cryptographic applications for our sampler, most of which use an integer sampler in one of three typical settings.

- The output distribution is the centered discrete Gaussian with fixed noise parameter. This is the case in most basic LWE based schemes, where the noise for the LWE instance is sampled using an integer sampler.

- The output distribution is the discrete Gaussian with fixed noise parameter, but varying center. This is the case in the online phase of Peikert's sampler [37]. In particular, if applied to q-ary lattices the centers are restricted to the set $\frac{1}{q}\mathbb{Z}$.
- The output distribution is the discrete Gaussian where both, the center and the noise parameter may vary for each sample. This is typically used as a subroutine for sampling from the discrete Gaussian over lattices, as the GPV sampler [22] or in the offline phase of Peikert's sampler.

The ideas presented in this work can be applied to any of these settings. In particular, the algorithms in Sect. 5 can be used to achieve new time-memory trade-offs in all three cases. The optimal trade-off is highly application specific and depends on a lot of factors, for example, the target platform (hardware vs. software), the cost of randomness (TRNGs vs. PRNGs), available memory, cost of evaluating $\exp(\cdot)$, cost of basic floating point/integer arithmetic, etc. In the following we present an experimental comparison of our algorithm to previous algorithms. Obviously, we are not able to take all factors into account, so we restrict ourselves to a comparison in a software implementation, where all algorithms use the same source of randomness (`NTL`'s PRNG), evaluate the randomness bit by bit in order to minimize randomness consumption, and use only elementary data types during the sampling. In particular, whenever FP arithmetic is necessary or $\rho_s(\cdot)$ needs to be evaluated during the sampling, all the algorithms use only double or extended double precision. This should be sufficient since we are targeting around 100 bits of security and the arguments in Sect. 3 apply to any algorithm. We do not claim that the implementation is optimal for any of the evaluated algorithms, but it should provide a fair comparison. We instantiated our algorithms with the parameters as listed at the end of Sect. 5.3. Our implementation makes no effort towards a constant-time implementation. Even though turning Algorithm 1 into a constant-time algorithm is conceptually simple (cf. Sect. 5.4), this still requires a substantial amount of design and implementation effort, which is out of the scope of this work.

When referring to specific settings of the parameter s, we will often refer to it as multiple of $\sqrt{2\pi}$. The reason is that two slightly different definitions of $\rho_s(\cdot)$ are common in the literature and the factor $\sqrt{2\pi}$ converts between them. While we found one of them to be more convenient in the analytic part of this work, most previous experimental studies [11, 39] use the other. So this notation is for easier comparability.

6.4 Fixed Centered Gaussian

In this section we consider the simplest scenario for discrete Gaussian sampling: sampling from the centered discrete Gaussian distribution above a certain noise level. This is accomplished by Algorithm 2. Note that the parameter s_0 allows for a time memory trade-off in our setting: the larger s_0, the more memory required by our base sampler (Knuth-Yao), but the fewer the levels of recursion. More precisely, the memory requirement grows linearly with s_0, while the running time decreases logarithmically.

We compare the method in different settings to the only other adjustable time-memory trade-off known to date – the discrete Ziggurat. For our evaluation we modified the implementation of [11] to use elementary data types only during the sampling (as opposed to arbitrary precision arithmetic in the original implementation). The baseline algorithms in this setting are the Bernoulli-type sampler and Karney's algorithm, as they allow to sample from the centered discrete Gaussian quite efficiently using very little or no memory. Figure 1 shows the result of our experimental analysis for a set of representative s's. We chose the examples mostly according to the examples in [11], where we skipped the data point at $s = 10\sqrt{2\pi}$, since this is already a very narrow distribution which can be efficiently sampled using Knuth-Yao with very moderate memory requirements. Instead, we show the results for $s = 2^{14}\sqrt{2\pi}$ (chosen somewhat arbitrarily), additionally to data points close to the ones presented in [11]: $s \in \{2^5, 2^{10}, 2^{17}\}\sqrt{2\pi}$.

Figure 1 shows that the two algorithms complement each other quite nicely: while Ziggurat allows for better trade-offs in the low memory regime, using convolution achieves much better running times in the high memory regime. This suggests that Ziggurat might be the better choice for constrained devices, but recall that it requires evaluations of $\exp(\cdot)$. So if s is not too large, even for constrained devices the convolution type sampler can be a better choice (see for example [39]).

Note that the improvement gained by using more memory deteriorates in our implementation, up to the point where using more memory actually hurts the running time (see Fig. 1, bottom right). A similar effect can be observed with the discrete Ziggurat algorithm. At first sight this might be counter-intuitive, but can be easily explained with a limited processor cache size: larger memory requirement in our case means fewer cache hits, which results in more RAM accesses, which are much slower. This nicely illustrates how dependent this trade-off is on the speed of the available memory. Since fast memory is usually much more expensive than slower memory, for a given budget it is very plausible that the money is better spent on limited amounts of fast memory and using Algorithm 2 rather than implementing the full Knuth-Yao with larger and slower memory. In our specific example (Fig. 1, bottom right), this means that using a convolution of two samples generated by smaller Knuth-Yao samplers is actually faster than generating the samples directly with a large Knuth-Yao sampler.

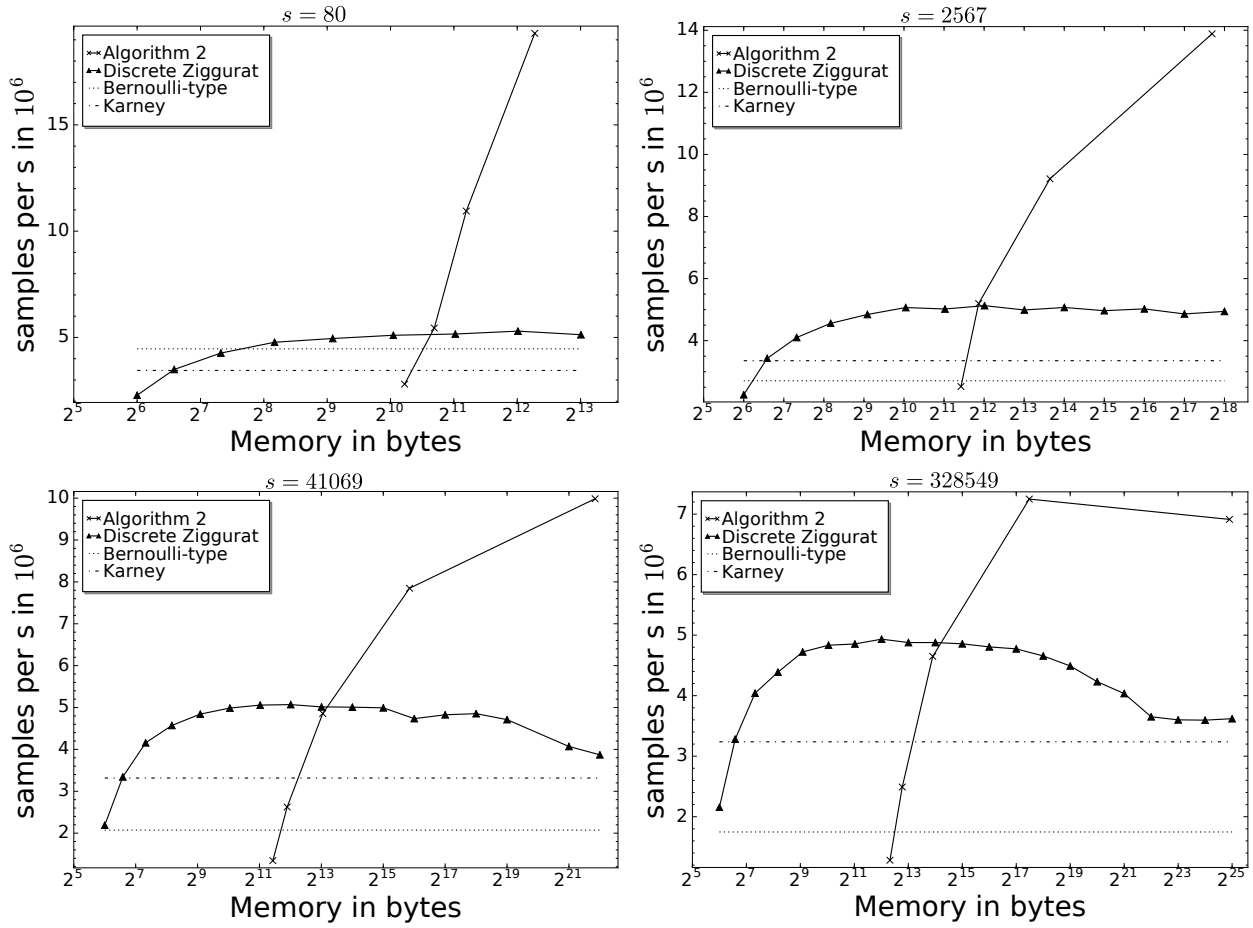

Figure 1: Time memory trade-off for Algorithm 2 and discrete Ziggurat compared to Bernoulli-type sampling and Karney's algorithm for $s \in \{2^5, 2^{10}, 2^{14}, 2^{17}\}\sqrt{2\pi}$. Knuth-Yao corresponds to right most point of Algorithm 2.

6.5 Fixed Gaussian with Varying Center

We now turn to the second setting, where the noise parameter is still fixed but the center may vary. In order to take advantage of the fact that the noise parameter is fixed and the center in a restricted set for the online phase, Peikert suggested that "if q is reasonably small it may be worthwhile (for faster rounding)

to precompute the tables of the cumulative distribution functions for all q possibilities" [37]. This might be feasible, but only for very small q and s (depending on the available memory). If not enough memory is available, there is currently no option other than falling back to Karney's algorithm or rejection sampling.

Depending on the cost of randomness, speed and amount of available memory and processor speed for arithmetic, Knuth-Yao can be significantly faster than Karney's algorithm. For example, in our prototype implementation, Knuth-Yao was up to 6 times faster, but keep in mind that this number is highly platform dependent and can vary widely. Accordingly, we can afford to invoke Knuth-Yao several times, sacrificing some running time for memory savings, and still outperform Karney's algorithm. Our algorithms offer exactly this kind of trade-off. There are two ways in which we can take advantage of convolution theorems to address the challenge of having to store q Knuth-Yao samplers. The first simply consists in storing the samplers for some smaller s_0, which will reduce the required memory by a factor s/s_0. After obtaining a sample from the right coset, using only the 0-coset we can generate and add a sample from a wider distribution to obtain the correct distribution. This is very similar to Algorithm 2 with the additional step of adding a sample from the right coset, where we simply invoke Corollary 4.1 once more. This step will increase the running time by at most $\log_{s_0} s$ additively (cf. Lemma 5.1).

Note that there is a limit to this technique though, since we need $s_0 > \sqrt{2}\eta_\epsilon(\mathbb{Z})$ for the convolution to yield the correct output distribution. If s is already small, but there is not enough memory available because q is too large, this approach will fail. In this case we can use the algorithm from Sect. 5.2 to reduce the number of samplers needed to be stored. In particular, for any base b such that[4] $\mathrm{rad}(q) \mid b$, we can cut down on the memory cost by a factor q/b, which will increase the running time by $\lceil \log_b q \rceil$. For this, we simply need to express the center c in the base b and round the digits individually using $\textsc{SampleC}_b$. For example, if q is a power of a small prime p, we can choose b to be any multiple of p. This can dramatically increase the modulus q for which we can sample fast with a given amount of memory, assuming $\mathrm{rad}(q)$ is small. As a more specific example, say q is a perfect square and let $b = \sqrt{q}$. Instead of storing q Knuth-Yao samplers and invoking one when a sample is required for a coset $\frac{1}{q}\mathbb{Z}$, we can store b samplers and randomly round each of the 2 digits of the center in base b successively. This effectively doubles the running time, but this is likely to still be much faster than Karney's algorithm (again, depending on the platform), but we reduced the amount of necessary memory by a factor $\sqrt{q}$.

Clearly, depending on the specific q, s and platform, the two techniques can be combined. The optimal trade-off depends on all three factors and has to be evaluated for each application. Our algorithms provide developers with the tools to optimize this trade-off and make the most of the available resources.

6.6 Varying Gaussian

Finally, we evaluate the practical performance of our full sampler, SampleZ. Precomputing the value K, as suggested in Sect. 5.3, made little difference in our software implementation and we show results for the algorithm that does not precompute K. The bottleneck in our algorithm is the call to SampleC, as it consumes a number of samples which depends on the base b. Again, similar to the previous section, the base b offers a time-memory trade-off, which is the target of our evaluation. We experimented with the sampler for a wide range of noise parameters s, but since our algorithm is essentially independent of s (as long as it is $\leq s_{\max}$), it is not surprising that the trade-off is essentially the same in all cases. Accordingly, we present only one exemplary result in Fig. 2. As a frame of reference, rejection sampling achieved $0.994 \cdot 10^6$ samples per second, which shows that by spending only very moderate amounts of memory ($<$ 1mb), our algorithm can match and outperform rejection sampling. On the other hand, Karney's algorithm achieved $3.281 \cdot 10^6$ samples per second, which seems out of reach for reasonable amounts of memory, making it the most efficient choice in this setting, if no other criteria are of concern. But we stress again that this depends highly on how efficiently Knuth-Yao can be implemented compared to Karney's algorithm on the target platform. While the running time of both, rejection sampling and Karney's algorithm depends on s, this dependence is rather weak (logarithmic with small constants) so the picture does not change much for other noise parameters.

[4] This is the condition for $\frac{1}{q}$ being expressible as a finite number in base b.

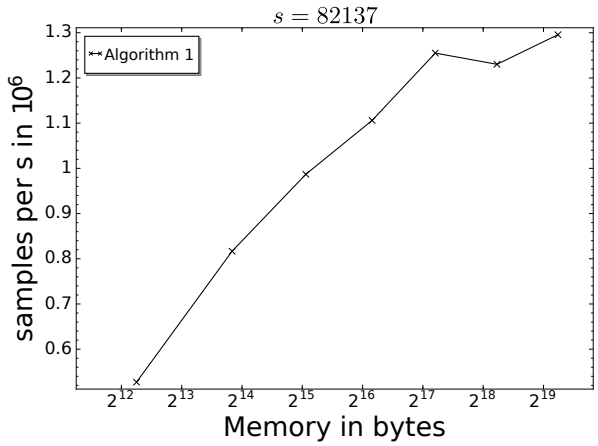

Figure 2: Time memory trade-off Algorithm 1 for $s = 2^{15}\sqrt{2\pi}$.

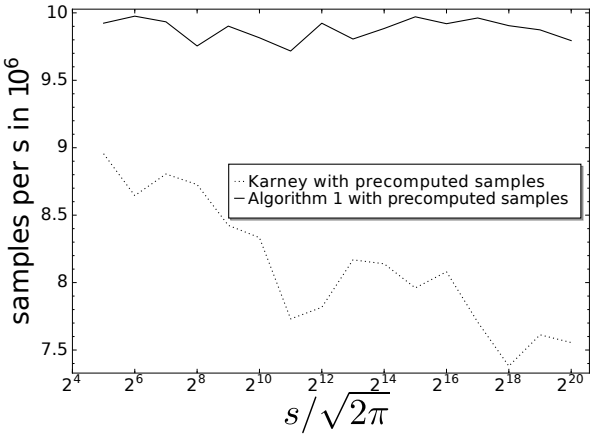

Figure 3: Performance of Algorithm 1 compared to Karney's algorithm, (online phase only).

Recall that our algorithm can be split into online and offline phase, since the base samples are independent of the target distribution. Karney's algorithm also initially samples from a Gaussian that is independent of the target distribution, so a similar approach can be applied. However, the trade-off is fixed and no speed-ups can be achieved by spending more memory.

We tested both algorithms, where we assumed that the offline phase is free, for a wide range of s. For this, we fixed $b = 16$ for our algorithm, which seemed to be a good choice in our setting. Note that similar to Sect. 6.4, spending more memory (and increasing b) should in theory only improve the algorithm. But if this comes at the cost of slowing down memory access due to a limited cache size, this can actually hurt performance. The results are depicted in Fig. 3. The graph allows for two interesting observations: First, our algorithm consistently outperforms Karney's algorithm in this setting. So if the offline phase can be considered to be free or a limited life-time is acceptable (cf. Sect. 5.4), our algorithm seems to be the better choice. Second, as expected, our algorithm is essentially independent of s (as long as it is $< s_{\max}$), while the performance of Karney's algorithm deteriorates as s grows. This is due to the fact that Karney's algorithm requires to sample a uniform number in $[0, s]$ during the online phase, which is logarithmic in s. This leads to a larger gap between the performance of the two algorithms as s grows, and supports the claim that our sampler allows for an efficient constant time implementation. In contrast, both Karney's algorithm and rejection sampling seem to be inherently costly to turn into constant time algorithms, due to their dependence on s and the fact that they are probabilistically rejecting samples.

In summary, we believe that there are a number of applications and target platforms, where our algorithm will be the best choice to implement a discrete Gaussian sampler.

A Distinguishing Distributions

In a recent work [44], Saarinen claimed that a sampling algorithm approximating a distribution using p-bit fixed point approximations, achieves about $2p$ bits of security, i.e. any algorithm successfully distinguishing the two distributions with constant advantage requires at least 2^{2p} running time. The claim is based on a theorem of Gregory and Paul Valiant [47], which, roughly speaking (and oversimplifying), presents a tester that can distinguish any unknown distribution from a known one using $O(1/\epsilon^2)$ samples, where ϵ is the statistical distance between the two distribution. Furthermore, [47] shows that this is tight: there is no algorithm that can distinguish *any* unknown distribution from a known one using fewer samples. Put differently, this result shows that for every distribution $\mathcal{P}$ *there exists* an unknown distribution $\mathcal{Q}$ with $\Delta_{SD}(\mathcal{P}, \mathcal{Q}) \leq \epsilon$ that requires $O(1/\epsilon^2)$ samples to distinguish from $\mathcal{P}$. The author of [44] seems to misinterpret

this to mean that for any known distributions $\mathcal{P}$, *any* unknown distribution $\mathcal{Q}$ with $\Delta_{SD}(\mathcal{P}, \mathcal{Q}) \leq \epsilon$ requires at least $O(1/\epsilon^2)$ samples to distinguish. The conclusion drawn in [44] is that a p-bit fixed point approximation is sufficient for $2p$ bits security, as it implies a statistical distance of $\lesssim 2^{-p}$.

Paradoxically, [44] also mentions a simple attack on such an algorithm which is dubbed *tail detector test*: consider an element x in the support of $\mathcal{P}$, such that $\mathcal{P}(x) \approx 2^{-p-2}$ is so small that its fixed point approximation is 0, i.e. x is not in the support of the approximate distribution $\mathcal{Q}$[5]. Simply drawing $\sim 2^{p+2}$ samples, one would expect x to appear once if drawing from $\mathcal{P}$, and not at all if drawing from $\mathcal{Q}$. This gives a simple way of distinguishing the two distributions in running time only $\sim 2^p$. In [44], this issue is stated but dismissed by conjecturing *"that lack of tail has only marginal effect on the entropy of random quantities and the security of the resulting cryptosystem."*

While this simple tail detector test is already enough to falsify the claim made in [44], we reinforce the point by presenting a simple generalization of the attack that also applies to elements that are in the support of both distributions but have very small probabilities.

Lemma A.1 *Let $\mathcal{P}$ and $\mathcal{Q}$ be two distributions and x an element in the support of $\mathcal{P}$ with $\epsilon = \mathcal{P}(x) - \mathcal{Q}(x)$. Then there exists an algorithm that, given oracle access to a distribution, can distinguish between $\mathcal{P}$ and $\mathcal{Q}$ with advantage $O(\exp(-\mathcal{Q}(x)/\epsilon))$ using $1/\epsilon$ samples in expectation.*

Proof: Let the Poisson distribution with parameter λ be $\mathcal{P}oi_\lambda(k) = \lambda^k e^{-\lambda}/k!$. The algorithm is based on the following observation: if we draw $n' \leftarrow \mathcal{P}oi_n$ and then draw n' samples independently from a distribution $\mathcal{P}$, the frequency of every element x in the support of $\mathcal{P}$ follows the Poisson distribution with parameter $n \cdot \mathcal{P}(x)$ [47]. Define $q = \mathcal{Q}(x)$. Consider an algorithm that draws $n' \leftarrow \mathcal{P}oi_{1/\epsilon}$, draws n' samples from the distribution and outputs 1 iff x is not in the set of samples. The probability that the algorithm outputs 1 when drawing from $\mathcal{Q}$ is $\mathcal{P}oi_{q/\epsilon}(0) = \exp(-q/\epsilon)$, and if drawing from $\mathcal{P}$ is $\mathcal{P}oi_{\mathcal{P}(x)/\epsilon}(0) = \exp(-\mathcal{P}(x)/\epsilon) = \exp(-(q+\epsilon)/\epsilon)$. The algorithm's distinguishing advantage is thus $\mathcal{P}oi_{q/\epsilon}(0) - \mathcal{P}oi_{\mathcal{P}(x)/\epsilon}(0) = (1 - \exp(-1))\exp(-q/\epsilon) = O(\exp(-q/\epsilon))$. □

Lemma A.1 shows that picking any element in the support of $\mathcal{P}$ with $\mathcal{P}(x) = O(2^{-p} \ln p)$, which is approximated using a p-bit fixed point number (i.e. $\epsilon \approx 2^{-p}$), yields a distinguishing attack in $\text{poly}(p) \cdot 2^p$. Note that the attack does not apply to floating point approximations, because they guarantee that $\mathcal{Q}(x)/\epsilon \geq 2^p$.

Acknowledgment

We thank the authors of [11] for providing the source code of their implementation of different discrete Gaussian samplers, Ilya Mironov for pointing out the connection between differential privacy and the max-log distance, and the CRYPTO'17 reviewers for helpful comments. This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contract number W911NF-15-C-0226.

References

[1] S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, 2010.

[2] S. Agrawal, D. Boneh, and X. Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *CRYPTO*, 2010.

[3] M. R. Albrecht, C. Cocis, F. Laguillaumie, and A. Langlois. Implementing candidate graded encoding schemes from ideal lattices. In *ASIACRYPT*, 2015.

[5] By nature of the distributions involved in lattice-based cryptography, such an element is very likely to exist.

[4] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange—a new hope. In *USENIX Security*, 2016.

[5] S. Bai, A. Langlois, T. Lepoint, D. Stehlé, and R. Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. In *ASIACRYPT*, 2015.

[6] X. Boyen. Attribute-based functional encryption on lattices. In *TCC*, 2013.

[7] X. Boyen and Q. Li. Attribute-based encryption for finite automata from LWE. In *ProvSec*, 2015.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[9] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In *CRYPTO*, 2011.

[10] Z. Brakerski, V. Vaikuntanathan, H. Wee, and D. Wichs. Obfuscating conjunctions under entropic ring LWE. In *ITCS*, 2016.

[11] J. Buchmann, D. Cabarcas, F. Göpfert, A. Hülsing, and P. Weiden. Discrete Ziggurat: A time-memory trade-off for sampling from a Gaussian distribution over the integers. In *SAC*, 2013.

[12] N. Döttling and J. Müller-Quade. Lossy codes and a new variant of the learning-with-errors problem. In *EUROCRYPT*, 2013.

[13] C. Du and G. Bai. Towards efficient discrete Gaussian sampling for lattice-based cryptography. In *FPL*, 2015.

[14] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal Gaussians. In *CRYPTO*, 2013.

[15] L. Ducas, V. Lyubashevsky, and T. Prest. Efficient identity-based encryption over NTRU lattices. In *ASIACRYPT*, 2014.

[16] L. Ducas and D. Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT*, 2015.

[17] L. Ducas and P. Q. Nguyen. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In *ASIACRYPT*, 2012.

[18] N. C. Dwarakanath and S. D. Galbraith. Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 2014.

[19] J. Folláth. Gaussian sampling in lattice based cryptography. *Tatra Mountains Mathematical Publications*, 2015.

[20] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[21] C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *EUROCRYPT*, 2011.

[22] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.

[23] L. Groot Bruinderink, A. Hülsing, T. Lange, and Y. Yarom. Flush, Gauss, and reload – a cache attack on the BLISS lattice-based signature scheme. In *CHES*, 2016.

[24] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O'Neill. On practical discrete Gaussian samplers for lattice-based cryptography. *IEEE Transactions on Computers*, 2016.

[25] C. F. F. Karney. Sampling exactly from the normal distribution. *ACM Trans. Math. Softw.*, Jan. 2016.

[26] A. Langlois, D. Stehlé, and R. Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. In *EUROCRYPT*, 2014.

[27] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, 2010.

[28] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

[29] D. Micciancio and C. Peikert. Hardness of SIS and LWE with small parameters. In *CRYPTO*, 2013.

[30] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measure. *SIAM Journal on Computing*, 2007. Preliminary version in FOCS 2004.

[31] D. Micciancio and M. Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. Cryptology ePrint Archive, Report 2017/259, 2017. `http://eprint.iacr.org/2017/259`.

[32] I. Mironov. Renyi differential privacy. `http://arxiv.org/abs/1702.07476`.

[33] S. More and R. Katti. Discrete Gaussian sampling for low-power devices. In *PACRIM*, 2015.

[34] P. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. *J. of Cryptology*, 2009. Preliminary version in EUROCRYPT 2006.

[35] NIST. Post-quantum crypto standardization - call for proposals announcement. `csrc.nist.gov/groups/ST/post-quantum-crypto/cfp-announce-dec2016.html`.

[36] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, 2009.

[37] C. Peikert. An efficient and parallel Gaussian sampler for lattices. In *CRYPTO*, 2010.

[38] P. Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In *INDOCRYPT*, 2016.

[39] T. Pöppelmann, L. Ducas, and T. Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In *CHES*, 2014.

[40] T. Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. Cryptology ePrint Archive, Report 2017/480, 2017. `http://eprint.iacr.org/2017/480`.

[41] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of ACM*, Sept. 2009. Preliminary version in STOC 2005.

[42] O. Regev. The learning with errors problem (invited survey). In *CCC*, 2010.

[43] S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Compact and side channel secure discrete Gaussian sampling. Cryptology ePrint Archive, Report 2014/591, 2014. `http://eprint.iacr.org/2014/591`.

[44] M.-J. O. Saarinen. Gaussian sampling precision in lattice cryptography. Cryptology ePrint Archive, Report 2015/953, 2015. `http://eprint.iacr.org/2015/953`.

[45] M.-J. O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering*, 2017.

[46] S. Sinha Roy, F. Vercauteren, and I. Verbauwhede. High precision discrete Gaussian sampling on FPGAs. In *SAC*, 2014.

[47] G. Valiant and P. Valiant. An automatic inequality prover and instance optimal identity testing. In *FOCS*, 2014.