# Sampling the Integers with Low Relative Error [*]

Michael Walter

IST Austria

`michael.walter@ist.ac.at`

## Abstract

Randomness is an essential part of any secure cryptosystem, but many constructions rely on distributions that are not uniform. This is particularly true for lattice based cryptosystems, which more often than not make use of discrete Gaussian distributions over the integers. For practical purposes it is crucial to evaluate the impact that approximation errors have on the security of a scheme to provide the best possible trade-off between security and performance. Recent years have seen surprising results allowing to use relatively low precision while maintaining high levels of security. A key insight in these results is that sampling a distribution with low *relative error* can provide very strong security guarantees. Since floating point numbers provide guarantees on the relative approximation error, they seem a suitable tool in this setting, but it is not obvious which sampling algorithms can actually profit from them. While previous works have shown that inversion sampling can be adapted to provide a low relative error (Pöppelmann *et al.*, CHES 2014; Prest, ASIACRYPT 2017), other works have called into question if this is possible for other sampling techniques (Zheng *et al.,*, Eprint report 2018/309). In this work, we consider all sampling algorithms that are popular in the cryptographic setting and analyze the relationship of floating point precision and the resulting relative error. We show that all of the algorithms either natively achieve a low relative error or can be adapted to do so.

## 1 Introduction

A key building block in many lattice based constructions is discrete Gaussian sampling over the integers. This is the distribution over the integers that is proportional to the standard continuous Gaussian distribution. Accordingly, it also has two parameters: a center $c$ and a noise parameter $\sigma$. Sampling this distribution is one of the more complex operations in many lattice-based schemes and can be challenging to implement efficiently and securely.[1]

Recent years have seen increasing interest in implementations of lattice-based cryptography, which may be attributed to two of its properties. First, lattice problems are believed to be resistant to attacks involving quantum computers. While building a large scale computer that would threaten most of the cryptography currently in use is still an open problem, progress in that area has already led to an enormous effort in the search for *post-quantum* cryptography and compelled NIST to initiate the Post Quantum Standardization process[2]. In this process NIST is seeking to identify post-quantum secure key exchange/key encapsulation mechanisms and signature schemes. Not surprisingly, a large share of the first round submissions can be classified as lattice-based schemes. When such schemes employ discrete Gaussian sampling, it is usually in a setting where the parameters of the Gaussian are fixed once and for all, i.e. after initialization the sampling algorithm does not accept any parameters and generates samples from one fixed distribution.

A second reason for the rising interest in implementations of lattice-based cryptography is its versatility. Beginning with the breakthrough result of Gentry [9] on fully homomorphic encryption, a number of other

---

[1]Technically, this distribution has infinite support, but it is folklore that the support can be truncated to size $O(\sigma)$ without hurting security, so in this entire work we consider the truncated version only.

[2]`https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization`

advanced primitives have been realized from lattices, like identity based or attribute based encryption. What sounded like futuristic science fiction 15 years ago, is now making its way into practical implementations [12, 4, 11]. When discrete Gaussian sampling is used in such implementations, it is almost always in the context of trapdoor sampling [10, 14, 8]. Here, the parameters of the distribution can change per query, i.e. for each query the algorithm expects the parameters as input.

The two settings make for different challenges. For examples, the simpler primitives mentioned first are often expected to run in constrained environments and to be side-channel resistant, both of which are challenging to achieve for discrete Gaussian sampling with reasonable performance. On the other hand, in this context the distribution is fixed, which allows for precomputation and several time-memory trade-offs. The challenge in the second setting is that the distribution changes per query, which makes it harder to obtain time-memory trade-offs to increase performance (although some results are known, e.g. [16, 1]). What the two settings have in common is that the distribution is usually approximated rather than sampled exactly. However, the corresponding security proofs usually assume the exact distribution. This naturally leads to the question how accurately the distribution needs to be approximated in order to maintain the security level. This is a crucial question since the quality of approximation has a large impact on the performance of the sampling algorithms. Classically, the statistical distance was used to evaluate this trade-off between approximation and security level. This lead to the common believe that a statistical distance of less than $2^{-k}$, achievable with $k$-bit fixed or floating point arithmetic, is required to maintain a security level of $k$ bits.

Surprisingly, a recent line of research [19, 3, 16, 20, 17] has shown that sampling a distribution with a relative error of $2^{-k}$ can preserve up to $2k$ bits of security, or more. For example, a typical theorem in this context can be informally summarized as follows.

**Theorem** (informal). *Let $S$ be a cryptographic primitive with access to a probability distribution $\mathcal{P}$, and let $\mathcal{Q}$ be a probability distribution such that the relative error between $\mathcal{P}$ and $\mathcal{Q}$ is bounded by $\delta_{\mathrm{RE}}(\mathcal{P}, \mathcal{Q}) \leq 2^{-\kappa/2}$. If $S$ is $\kappa$-bit secure, then $S$ is still $(\kappa - O(1))$-bit secure if $\mathcal{P}$ is replaced with $\mathcal{Q}$.*

Since floating point numbers with $k$ bits guarantee a relative approximation error of less than $2^{-k}$, this suggests that one can use $k$-bit floating point numbers to achieve security levels as large as $2k$ or more, as demonstrated in [19, 20]. In many settings this can lead to a significant improvement, much larger than a factor two, over the classical approach of using approximations of about $k = \lambda$ bits of precision to achieve $\lambda$ bits of security. This is because the common IEEE (extended) double precision standard with its 53 (64, resp.) bits of precision now allows for meaningful security guarantees. Using arbitrary precision libraries instead would be orders of magnitude slower than arithmetic with the common data types, which is often supported in hardware. Previous works, such as [19, 20, 16], make use of different measures between distributions, e.g. Rényi divergences of different orders, and achieve some stronger results under certain conditions then abovementioned theorem. We point out that the starting point of these analyses is still the relative error between distributions, which is then used to prove bounds on other measures. We are not aware of any analysis of this type that is not based on the relative error.

Due to the guarantees provided by floating point numbers, it is tempting to assume that simply approximating all probabilities/numbers in your favorite sampling algorithm using $k$-bit floating point numbers is sufficient to preserve $2k$ bits of security. Unfortunately, this is not true in general. For example, [19] already pointed out that blindly using FP approximations in a sampling algorithm (in this case inversion sampling applied to the discrete Gaussian) can lead to catastrophic errors. A recent work [24] demonstrated that there are distributions for which one cannot even find a set of floating point numbers that approximate all the probabilities of the given distribution with small relative error and sum to 1 (thus representing a probability distribution). The authors claim that their result implies that sampling with small (in the sense needed for the results in [19, 3, 16, 20, 17] to apply) floating point error is impossible using "methods such as rejection sampling". On the other hand, this question has been addressed in the context of inversion (CDT) sampling [19, 20]. The proposed algorithms use $k$-bit floating point numbers (during the online phase) only and the resulting relative error of the distribution is only $O(n2^{-k})$, where $n$ is the number of elements in the support. The key point here is that at least in the case of inversion sampling, storing all probabilities of a distribution approximating the desired one is not necessary. This naturally raises the question if this is also the case for other samplers that are popular in the cryptographic context. This strikes us as a very

important question for practical implementations of schemes like, for example, some of the ones submitted to the NIST Post Quantum Standardization process or the PALISADE project[3], due to the large potential improvement. This is amplified by the fact that the performance of sampling algorithms can vary widely on different platforms due to different time-memory-randomness trade-offs they achieve. So different samplers will be more suitable in different settings and it is useful to investigate, which samplers provide a low relative error and thus achieve strong security guarantees.

**Overview of Samplers**   In this work, we will survey several samplers and show how to adapt them (if necessary) in order to ensure that the relative error does not significantly outgrow $2^{-k}$ when working with $k$-bit floating point numbers. To the best of our knowledge our survey includes all samplers that are popular and/or useful in context of discrete Gaussian sampling.

The first and simplest algorithm that has been used for discrete Gaussian sampling is plain *rejection sampling* [10]. In this algorithm, a sample from the support is chosen uniformly at random and accepted with probability proportional to the desired distribution. This algorithm is very simple and generic and requires little memory, but it is not very suitable for discrete Gaussian sampling due to a high rejection rate and the requirement to compute irrational probabilities. Nonetheless, we consider it first due to its generality and our observations will be useful in later sections.

In settings where the distribution is fixed and known in advance, there are a number of generic algorithms that allow for faster sampling by performing distribution dependent precomputation and using additional memory. Probably the most popular one in the context of discrete Gaussian sampling is *inversion (CDT) sampling* [18, 19]. Inversion sampling requires to precompute and store the cumulative distribution table, after which the sampling step becomes a binary search for a random element in this table. While this is much faster than rejection sampling, it requires a large amount of memory if the desired distribution has a large support (i.e. the noise parameter $\sigma$ is very large).

Less well known in the cryptographic community seems to be the *alias method*, but we include it in our survey, because we believe it deserves more attention. The alias method requires a little more precomputation, but the memory is roughly the same as for inversion sampling. In contrast to inversion sampling, the online phase only requires a Bernoulli trial using a bias randomly chosen from a precomputed set. This has the potential of being even faster and easier to implement securely (e.g. in constant time) than inversion sampling.

A third algorithm along similar lines is *Knuth-Yao sampling* [7, 21]. It also requires to precompute the probabilities from the entire support of the distribution. To generate a sample, it traverses a tree structure, which is based on the binary expansion of these probabilities. This tree structure can either be precomputed explicitly, which requires an even larger memory overhead than inversion or alias sampling, or locally on-the-fly, in which case the memory requirement is equivalent to that of the previous two methods. However, the latter is also naturally less efficient. Knuth-Yao sampling is often recommended for environments where randomness is expensive, since it is provably nearly randomness optimal.

The three generic algorithms discussed above all have the advantage that they avoid any arithmetic with reals during the online phase, which seems to make them suitable for constraint environments. Unfortunately, they all suffer from memory requirements that can be prohibitive, especially on constrained devices. To mitigate this, [5] proposed an algorithm (which we denote by the *BLISS sampler*) that provides a different time-memory trade-off. This algorithm cleverly tailors rejection sampling to the discrete Gaussian to reduce the rejection rate and breaks the rejection step into a logarithmic number of Bernoulli trials with precomputed bias. Accordingly, it requires only logarithmic memory and no rational arithmetic, which makes it much more suitable for constrained devices.

With the exception of the rather slow rejection sampling, all algorithms described so far only allow to sample from a distribution that is fixed and known in advance, since they require distribution dependent precomputation. In more advanced settings, where the distribution is not fixed in advance, one is left with much fewer options. One of these options is Karney's algorithm [13]. It resembles the BLISS sampler insofar as it is also a rejection sampler that is tailored to the discrete Gaussian (albeit in a different way), but it is

---

[3]https://git.njit.edu/palisade/PALISADE

also able to perform the rejection step using no precomputation and using only integer arithmetic, assuming the parameters of the distribution are given as rationals. As such, it performs well in settings where the parameters of the desired distribution may vary per query [16].

**Contribution**  We distinguish between generic samplers that are not specific to discrete Gaussian sampling and algorithms that are specialized to discrete Gaussians.

We start with generic samplers and first make a rather simple observation that shows that rejection sampling does not require any modification and its output distribution will have small relative error if used with floating point numbers. Results from [19, 16] then show that the error in terms of KL divergence is only the square of the relative error. This directly contradicts the aforementioned claim made in [24]. We also include a section about inversion sampling and demonstrate that the solution provided in [19] applies more generally than only to discrete Gaussians. Then we show that alias sampling can be easily adapted to achieve small relative error with almost no overhead. In Section 2.4, we consider a specific version of Knuth-Yao sampling and show that there are different ways to adapt it leading to different trade-offs in output quality versus running time. We note that we demonstrate the result on a specific version for concreteness and that the results easily carry over to other implementations.

We will then move on to the two specialized algorithms. We first address the BLISS sampler, where we show that above observation w.r.t. rejection sampling easily implies similar results for the BLISS sampler. Finally, we will consider Karney's algorithm, which is originally an exact algorithm, but for implementation purposes it is convenient to consider a floating point version of it, e.g. in the context of trapdoor sampling. While the observation about rejection sampling is useful also in this context, Karney's algorithm requires much more work to convert to a correct floating point algorithm due to some arithmetic that is required during the online phase. This part contains the bulk of our technical work. We remark that our floating point version introduces a lot of new conditionals to guard against numerical errors, which can hurt performance on standard CPUs significantly due to its detrimental effect on pipelining. To evaluate this effect and compare our algorithm to a high precision variant, we provide an implementation of our algorithm. Our experimental results on a standard PC suggest that at least for such architectures the slow down introduced by the conditionals is very acceptable compared to the speed up achievable in comparison with a high precision variant of Karney's algorithm. However, note that the performance of samplers depends strongly on the platform characteristics and the final choice should be made on a case-by-case consideration with a specific target platform in mind. We will release our implementation to the public.

**Applications**  The algorithms considered in this work can be used as building blocks in many kinds of cryptographic primitives. The results shown here on the relative error can be combined with results from [19, 3, 16, 20, 17] to prove strong security (since [17] even for distinguishing primitives like encryption). Furthermore, the approximate samplers as surveyed in this work can be used as a basis for convolution samplers [19, 16]. These were first introduced to mitigate the large memory overhead of generic sampling techniques [5] and then shown to also yield a reduction from variable sampling to the fixed setting [16]. Such constructions ensure low relative error if instantiated with base samplers with low relative error. In this context, the results on alias sampling, inversion sampling and Knuth-Yao are particularly relevant, as their characteristics are very desirable for convolution sampling.

**Previous Work**  In this work we carry out some basic numerical analysis of some generic discrete sampling algorithms with a focus on the resulting relative error. Most previous work in the cryptographic context has focused on ensuring small statistical distances [6, 22, 5], or only considered the special case of inversion sampling [20, 19]. More specifically, [19] shows that by reordering the probabilities, inversion sampling is able to achieve a bounded relative error at least in the special case of discrete Gaussians. [20] presents a version of inversion sampling – conditional CDT sampling – that works for any discrete distribution and achieves similar bounds but is less efficient, since it requires multiple iterations per sample.

The work of [3] analyzed the BLISS sampler using Rényi divergences, and specifically the divergence of order infinity is closely related to the relative error discussed in this work. We will briefly revisit this

analysis in Section 3.1. Finally, a very recent work [23] showed that one can perform the rejection step in the BLISS sampler using polynomial approximations, which allows for much easier and faster constant-time implementation.

**Preliminaries**  In this work we make heavy use of the notation $(a \pm b)$ for $a, b \in \mathbb{R}$. By this we mean the interval $[\min(a + b, a - b), \max(a + b, a - b)]$. We will also do arithmetic with intervals and write $A \circ B$ for intervals $A, B$ and $\circ \in \{+, -, \cdot, /\}$. We define this to be the interval $[\min_{\alpha \in A, \beta \in B}(\alpha \circ \beta), \max_{\alpha \in A, \beta \in B}(\alpha \circ \beta)]$. Finally, we define arithmetic of intervals with reals by viewing any constant $c \in \mathbb{R}$ as the interval $[c, c]$.

A $k$-bit floating point (FP) approximation $\widehat{x}$ of a real $x$ stores the $k$ most significant bits of $x$ as the mantissa $m \in \mathbb{Z}_{2^k}$ together with a binary exponent $e \in \mathbb{Z}$ and a sign $s \in \{-1, 1\}$. The value of an FP number $(s, m, e)$ is $sm2^e$. This guarantees that the relative error is bounded by $\delta_{\mathrm{RE}}(x, \widehat{x}) = |x - \widehat{x}|/|x| \leq 2^{-k}$, or, equivalently, that $\widehat{x} \in (1 \pm 2^{-k})x$. For simplicity, we assume that there is no limit on the size of the exponent, i.e. there are no overflows or underflows. The standard guarantee provided by most FP systems is that an arithmetic operation on two FP numbers yields the closest FP number to the true result. This generally holds for the four basic operations $\{+, -, \cdot, /\}$. Two values that already contain an approximation error, the operations $\{\cdot, /\}$ do not increase the error in the result by much, while the operations $\{+, -\}$ can have a very detrimental effect on the approximation error.

We use calligraphic letters to denote distributions. The Bernoulli distribution with bias $p$ is denoted by $\mathcal{B}_p$. 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 $\mathcal{P}$.

In this note, we will be interested in samplers that use only FP numbers of a given precision $k$ (during the online phase) and output a distribution $\widehat{\mathcal{D}}$ such that $\delta_{\mathrm{RE}}(\mathcal{D}, \widehat{\mathcal{D}}) \leq O(2^{-k})$. Note that this necessarily implies that $\widehat{\mathcal{D}}$ has the same support as the original distribution $\mathcal{D}$.

# 2 Generic Samplers

Not surprisingly, the literature on discrete Gaussian sampling borrows heavily from known generic sampling techniques. In the following we analyze some popular sampling methods. The results of this section, while technically simple, could be of interest in other contexts than discrete Gaussian sampling.

## 2.1 Rejection Sampling

The following lemma describes the basic principle of rejection sampling specialized to the setting of discrete probability distributions, which we prove for completeness.

**Lemma 1** (Rejection Sampling). *Let $\mathcal{P}$ be a discrete probability distribution, called the* source distribution, *with support $S_{\mathcal{P}}$ and $f : S \mapsto \mathbb{R}_+$ a function with domain $S \subset S_{\mathcal{P}}$ defining the* target distribution *$\mathcal{Q}(x) = \frac{f(x)}{\sum_{x \in S} f(x)}$. Let $M \geq \max_{x \in S} \frac{f(x)}{\mathcal{P}(x)}$. Given a sampler for $\mathcal{P}$ it is easy to generate a sample from $\mathcal{Q}$ by repeatedly drawing a sample $x \leftarrow \mathcal{P}$ and accepting it with probability $\frac{f(x)}{M\mathcal{P}(x)}$, until the first sample is accepted.*

*Proof.* Accepting a sample $x$ with probability $\frac{f(x)}{M\mathcal{P}(x)}$ is equivalent to generating a uniform value $u \in [0, M\mathcal{P}(x)]$ and accepting $x$ if $u \leq f(x)$. In other words, this generates a pair $(x, u) \in S \times \mathbb{R}_+$ s.t. $u \leq M\mathcal{P}(x)$, which is distributed uniformly random among such pairs. Accepting $x$ only if $u \leq f(x)$ amounts to a Monte-Carlo method of producing a pair $(x, u) \in S \times \mathbb{R}_+$ s.t. $u \leq f(x)$ uniformly at random and thus the marginal relative probability of $x$ is exactly $f(x)$. $\qquad \square$

A particularly simple case is when $\mathcal{P}$ is the uniform distribution and $f(x) \leq 1$ for all $x \in S$. Then $M$ can be chosen as $|S_{\mathcal{P}}|$, which means the acceptance probability is simply $f(x)$ for each $x$ and no further floating point operation is necessary.

We now observe that rejection sampling naturally allows for FP approximations.

**Lemma 2.** *Let $f, \widehat{f} : S \mapsto \mathbb{R}_+$ be two functions with domain $S$ and let $\mathcal{D}, \widehat{\mathcal{D}}$ be the corresponding distributions obtained by normalization. If $\delta_{\mathrm{RE}}(f(x), \widehat{f}(x)) \leq \mu$ for all $x \in S$, then $\delta_{\mathrm{RE}}(\mathcal{D}, \widehat{\mathcal{D}}) \leq 2\mu + O(\mu^2)$.*

*Proof.* It is easy to see that $\sum_{x \in S} \widehat{f}(x) \in (1 \pm \mu) \sum_{x \in S} f(x)$. So for any $x \in S$ we have

$$\widehat{\mathcal{D}}(x) = \frac{\widehat{f}(x)}{\sum_{x \in S} \widehat{f}(x)} \in \frac{(1 \pm \mu)f(x)}{(1 \pm \mu) \sum_{x \in S} f(x)} = (1 \pm (2\mu + O(\mu^2)))\mathcal{D}(x).$$

$\square$

Combining the two lemmas easily shows that at least the simple version of rejection sampling as described above can be safely used with FP approximations.

**Remark 1.** *An important observation is that approximating the acceptance probability $\frac{f(x)}{M\mathcal{P}(x)}$ with a certain relative error is equivalent to approximating $f$ with the same relative error. This makes Lemma 2 also applicable to algorithms that approximate the acceptance probability. I.e., if two rejection samplers use the same source distribution and only differ by a relative error $\leq \mu$ in the acceptance probability for all $x$, their output distributions will have relative error of at most $2\mu + O(\mu^2)$. Implications of this will be discussed in Section 3.*

## 2.2 Inversion Sampling

While rejection sampling is very simple and versatile, for many distributions it can be rather slow in its simple form due to a large rejection rate. If the distribution is known in advance and has moderately sized support, one can use inversion sampling as a way to trade precomputation and memory for faster performance. For a discrete distribution $\mathcal{P}$ one computes the cumulative distribution table (CDT) of the desired distribution and stores it. The CDT consists of $n + 1$ numbers (where $n$ is size of the support) $0 = T[0] < T[1] < \cdots < T[n] = 1$ such that $\mathcal{P}(i) = T[i] - T[i-1]$ for all $i \in [n]$. When queried for a sample, a number $u \in [0, 1)$ is drawn uniformly at random and the smallest value $i$ with $T[i] \geq u$ is output. Clearly, the probability for any output $i$ is $\Pr[u \leq T[i] \wedge u > T[i-1]] = T[i] - T[i-1] = \mathcal{P}(i)$. The question that we are concerned with is whether a bound can put on the relative error of the output distribution when storing $k$-bit FP approximations of the values $T[i]$. In [19] it was shown that in the special case of discrete Gaussians, the error can grow arbitrarily when applying inversion sampling in the obvious way, but is bounded by $O(n2^{-k})$ when reversing the order of the probabilities. We now demonstrate that this is in fact a very general technique that works for any finite discrete distribution.

**Lemma 3.** *Let $\mathcal{P}$ be discrete probability distribution of $n$ elements with $\mathcal{P}(1) \leq \mathcal{P}(2) \leq \cdots \leq \mathcal{P}(n)$. Let $\widehat{\mathcal{P}}$ be the distribution output by an inversion sampler that approximates the entries in the CDT using $k$-bit FP numbers. Then $\delta_{\mathrm{RE}}(\mathcal{P}, \widehat{\mathcal{P}}) \leq n2^{-k+1}$.*

*Proof.* The inversion sampler stores the values $\widehat{T[i]} \in (1 \pm \mu)T[i]$, where $\mu = 2^{-k}$. So

$$
\begin{aligned}
\frac{\widehat{\mathcal{P}}(i)}{\mathcal{P}(i)} &= \frac{\widehat{T}[i] - \widehat{T}[i-1]}{\mathcal{P}(i)} \\
&\in \frac{(1 \pm \mu)T[i] - (1 \pm \mu)T[i]}{\mathcal{P}(i)} \\
&= \frac{(1 \pm \mu)\sum_{j \leq i} \mathcal{P}(j) - (1 \pm \mu)\sum_{j < i} \mathcal{P}(j)}{\mathcal{P}(i)} \\
&= 1 \pm \left( \frac{\mu \sum_{j \leq i} \mathcal{P}(j)}{\mathcal{P}(i)} + \frac{\mu \sum_{j < i} \mathcal{P}(j)}{\mathcal{P}(i)} \right).
\end{aligned}
$$

Due to the ordering of the probabilities we have $\mathcal{P}(j) \leq \mathcal{P}(i)$ for all $j \leq i$ and so the result follows. $\qquad\square$

Since inversion sampling as described above is only useful for distributions with polynomial support, the lemma shows that we can use inversion sampling for any such discrete distribution and achieve a small relative error by ordering the elements in the support according to their probabilities.

**Application to Bernoulli Sampling** Assume we are given a bias $p$ and want to sample from the respective Bernoulli distribution, i.e. the distribution over $\{0, 1\}$ with $p_0 = p$ and $p_1 = 1 - p$. The straight forward way of implementing this is to store $p$ and during the online phase one draws a uniform number $u$ in $[0, 1)$ and outputs 0 iff $u \leq p$. It should be clear that this can be viewed as a special case of inversion sampling with two element, i.e. $n = 2$. Applying the result from above shows that we can perform Bernoulli sampling and preserve a relative error of $2^{-k+2}$ if we reorder the probabilities. In this special case, this means we store an approximation of $p$ or $1 - p$, whichever one is smaller. In fact, a direct calculation shows that this results in a relative error of at most $2^{-k}$.

We also remark that [3] already observed that in the context of Bernoulli sampling one can assume w.l.o.g. that $p \leq \frac{1}{2}$ and that in this case an FP-approximation of $p$ results in a small Rényi divergence of order infinity, which is essentially equivalent to the relative error.

## 2.3 Alias Method

Another popular method of using precomputation to speed up sampling for a known distribution is the alias method. Let $\mathcal{P}$ be a given probability distribution with support $S_{\mathcal{P}}$, where $|S_{\mathcal{P}}| = n$. The idea of alias sampling is to reduce sampling from $\mathcal{P}$ to sampling from a Bernoulli sampler randomly chosen from a carefully crafted set of Bernoulli samplers. To get an intuitive understanding of the algorithm, picture a set of buckets, one for each element in $S_{\mathcal{P}}$. These buckets have size $\frac{1}{n}$ and are filled with the probability mass of the corresponding element. Of course, some buckets are not full, while others are overflowing. In order to smooth things out, pick a bucket $i$ that is not full, i.e. assume $p_i < \frac{1}{n}$. Mark its current filling level and then fill it up by moving probability mass from some overfull bucket, for example bucket $j$ (where we assume $p_j > \frac{1}{n}$). The top of the now exactly full bucket $i$ is labeled with $j$ (so $j$ is $i$'s *alias*). Note that bucket $j$ might now still be overflowing or not be full. Continue this procedure with a different $i'$ and $j'$ until every bucket is filled to exactly $\frac{1}{n}$. This is the offline/precomputation phase. In order to sample from $\mathcal{P}$, we select one of the buckets uniformly at random. Say we selected bucket $i$, which has a mark at $p_i$ and alias $j$. We now draw a number $r$ uniformly at random in $[0, \frac{1}{n})$. If $r \leq p_i$, we output $i$, otherwise we output the alias $j$. This is equivalent to drawing a Bernoulli sample with bias $np_i$ and outputting $i$ if the sample is 1 and $j$ otherwise. The reason this works is that we did not change the amount of probability mass corresponding to a value. E.g., in the example of $i$ and $j$ above, assume that after moving mass from bucket $j$ to bucket $i$, both buckets happen to be full. (In particular, this means that $j$ does not have an alias and when selecting bucket $j$ during sampling, we always output $j$.) Then the probability of obtaining $i$ as a sample is exactly $\frac{1}{n}np_i = p_i$ and for $j$ it is $\frac{1}{n}(1 + 1 - np_i) = \frac{2}{n} - p_i$. Since we assume bucket $j$ was exactly full after moving $\frac{1}{n} - p_i$ mass to bucket $i$, we have $p_j = \frac{2}{n} - p_i$, which shows that alias sampling works correctly here. In

summary, the alias method works by constructing and storing $n$ Bernoulli samplers $B_j$ in the offline phase. In the online phase, one of the samplers is selected uniformly at random, a sample is obtained and depending on the result, one of two values is output. In our setting we assume that the bias of each sampler is computed exactly during the offline phase and the Bernoulli sampler is then approximated as described in Section 2.2.

**Lemma 4.** *The alias method can be implemented using only $k$-bit FP numbers during its online phase and guarantee a relative error of $2^{-k}$.*

*Proof.* In the following we view the Bernoulli samplers as directly outputting one of the two values in the table, i.e. let $x_j, y_j \in [n]$ be the two values associated with the $j$-th Bernoulli sampler via the alias table and let $b_j$ be the bias of that sampler. Then define

$$
p_{B_j}(i) = \begin{cases} b_j & \text{if } i = x_j \\ 1 - b_j & \text{if } i = y_j \\ 0 & \text{otherwise.} \end{cases}
$$

The probability that a value $i$ is output by the alias sampler is then $p_i = \frac{1}{n} \sum_{j=1}^{n} p_{B_j}(i)$. From Section 2.2 we know that we can sample each Bernoulli sampler up to a relative error of $\mu = 2^{-k}$, so the probability of $i$ under the approximate sampler is $\widehat{p}_i \in \frac{1}{n} \sum_{j=1}^{n} (1 \pm \mu) p_{B_j}(i) = \frac{(1 \pm \mu)}{n} \sum_{j=1}^{n} p_{B_j}(i)$, which shows that the relative error is at most $2^{-k}$. $\qquad\square$

It is noteworthy that alias sampling actually allows to achieve the optimal bound of $2^{-k}$ for the relative error using $k$-bit FP numbers only. This is in contrast to inversion sampling, where the relative error can be larger by a factor $O(n)$.

## 2.4 Knuth-Yao Sampling

Yet another way to spend precomputation and storage in order to speed up sampling from a known distribution is Knuth-Yao sampling. Knuth-Yao sampling constructs a binary tree, known as the distribution generating tree (DGT), during initialization from the given probabilities $p_i$. During the online phase, a random path from the root is traversed to the end, at which point the label of the leaf determines the sample that is output. Note that a leaf on level $t$ of the tree has a probability of $2^{-t}$ of being sampled. So if for a sample $i$ there are $l$ leafs, where each is on a distinct level $t_j$, then the probability of outputting $i$ is exactly $\sum_{j}^{l} 2^{-t_j}$. The tree is constructed from $p_i$ such that for every '1' in position $t$ in the fixed point binary expansion of $p_i$ there is a leaf for $i$ on level $t$. Then it is easy to see that the probability of $i$ is exactly $p_i$. The fact that $\sum_i p_i = 1$ ensures that the tree can be easily constructed from the binary representation of the $p_i$ and that there is a label for every leaf.

Knuth-Yao is often praised for its randomness efficiency, which is close to the entropy of the distribution and thus close to optimal. We point out though that at least in the context of discrete Gaussians, which have entropy[4] $\sim \log \sigma$, both inversion sampling and alias sampling can also be implemented to only require $\log n + O(1) = \log \sigma + O(1)$ random bits.

There are several ways to implement Knuth-Yao:

1. Construct the tree structure and traverse it as described above. This incurs a significant memory overhead, so this version seems only applicable if memory is not an issue.

2. In order to reduce the memory overhead, one can construct each level of the tree on-the-fly during the traversal using a table [7]. This table requires less storage than the tree itself, but still leaves a significant overhead. (In essence, the table contains a number in $[n]$ for every '1' in the binary expansions of the $p_i$, which means this table still requires larger storage than the probabilities itself by a factor $O(\log n)$.)

---

[4]Here, $\sigma$ is the noise parameter of the discrete Gaussian. See Definition 1.

3. Finally, [21] introduced a variant that constructs the tree entirely on-the-fly only using the probability table. In this case the storage requirement essentially matches the one of inversion sampling and alias sampling.

Floating point numbers can be used to reduce the storage requirement in all three variants. When using FP approximations $\widehat{p}_i$ of $p_i$ for all $i$, the difficulty is to deal with the fact that $\sum_i \widehat{p}_i \neq 1$. Note that we can easily ensure that $\sum_i \widehat{p}_i \leq 1$ by requiring that $\widehat{p}_i \leq p_i$ by always rounding down the approximations, which will increase the relative error by at most a factor 2. In this case, the DGT tree is not well defined and the result is that the algorithm might not ever encounter a leaf. However, this case is easily detected since the depth of the tree is bounded by $k - \min \log \widehat{p}_i$, where $k$ is the precision of the FP numbers. In that case, one can simply restart the algorithm, which essentially turns this algorithm into a rejection sampler. We demonstrate this approach in Algorithm 1, where the subroutine $\text{BIT}(p = (s, m, e), c)$ gives access to the $c$'th bit of the fixed point representation of the number represented by the $k$-bit FP number $p$.

---

**Algorithm 1** Floating point version of Knuth-Yao, adapted from [21]. The list $(p_i)_i$ consists of $k$-bit FP numbers.

| | |
|---|---|
| $\text{SAMPLEKY}((p_i)_i, k)$ | $\text{BIT}((p = (s, m, e), c)$ |
| 1     $c_{\max} = \lceil -\log \min_i p_i \rceil + k$ | 1     **if** $c < -e - k$ or $c > -e$ |
| 2     $d \leftarrow 0$ | 2       **return** $0$ |
| 3     **while** true | 3     **return** $m[c + e + k]$ |
| 4      **for** $c = 0$ to $c_{\max}$ | |
| 5       $b \leftarrow \{0, 1\}$ | |
| 6       $d \leftarrow 2d + 1 - b$ | |
| 7       **for** $r = n$ down to $0$ | |
| 8        $d \leftarrow d - \text{BIT}(p_r, c)$ | |
| 9        **if** $d = -1$ | |
| 10         **return** $r$ | |

---

**Theorem 1.** *Let $\mathcal{P}$ be a discrete distribution over a finite set with probabilities $p_i$. If given $k \in \mathbb{Z}$ and a list of $k$-bit FP numbers $(\widehat{p}_i)_i$ such that $\widehat{p}_i \leq p_i$ and $\delta_{\text{RE}}(p_i, \widehat{p}_i) \leq \mu$ for all $i$, then the output distribution $\widehat{\mathcal{P}}$ of Algorithm 1 satisfies $\delta_{\text{RE}}(\mathcal{P}, \widehat{\mathcal{P}}) \leq 2\mu + (O(\mu^2))$. Furthermore, the expected running time and expected randomness consumption is $(1 + \mu)H(\mathcal{P}) + O(1)$.*

*Proof.* Since Algorithm 1 corresponds to a randomized walk on a truncated DGT (see [21] for details), the relative probability of each sample $i$ is $\widehat{p}_i$. Now the first part of the theorem follows from Lemma 2.

For the second part notice that Algorithm 1 behaves, up to rejection, exactly like an exact implementation of Knuth-Yao, which has running time and randomness consumption $H(\mathcal{P})$. The result follows from the fact that the rejection probability is $\leq \mu$. □

**Remark 2.** *If the rejection step is undesirable for some reason, there is another way to deal with the situation where no leaf is encountered: simply always output $j = \arg\max_i p_i$. Note that $p_j > 1/n$ and so*

$$\widehat{p}_j + \mu \in (1 \pm \mu)p_j + \mu \in \left(1 \pm \left(\mu + \frac{\mu}{p_j}\right)\right) p_j \in (1 \pm (\mu + n\mu))p_j.$$

*This shows that this solution leads to a relative error of $\leq 2n\mu$.*

# 3 Specialized Algorithms

We now consider two algorithms that are specialized to discrete Gaussians. They are both based on rejection sampling and differ in the source distribution and the way the rejection step is carried out to achieve high efficiency. We start by defining the distribution we are interested in.

**Definition 1.** *We denote by $\mathcal{D}_{\mathbb{Z},\sigma,c}$ the distribution over $\mathbb{Z}$ that is proportional to $\rho_{\sigma,c}(x) = \exp(-\frac{(x-c)^2}{2\sigma^2})$. Furthermore, we define the distribution $\mathcal{D}_{\mathbb{Z},\sigma,c}^{+}$ to be the distribution proportional to $\mathcal{D}_{\mathbb{Z},\sigma,c}$ restricted to non-negative integers $\mathbb{Z}_{\geq 0}$. In either case, if the parameter $c$ is omitted, it is understood to be $0$.*

Note that sampling $\mathcal{D}_{\mathbb{Z},\sigma}$ is easily reduced to sampling $\mathcal{D}_{\mathbb{Z},\sigma}^{+}$: draw a sample $x \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{+}$ and a uniform bit $s \leftarrow \mathcal{U}(\{0,1\})$. If $x = s = 0$, reject $x$ and try again. Otherwise, return $(-1)^s \cdot x$. It is folklore that for cryptographic purposes one can view $\mathcal{D}_{\mathbb{Z},\sigma,c}$ as a $[c - t\sigma, c + t\sigma]$ bounded distribution for some constant $t$ due to its exponential decay in the tails. This follows from known bounds on the so called *smoothing parameter* [15] and a simple hybrid argument. Furthermore, from an algorithmic perspective we can assume w.l.o.g. that $c \in [0, 1)$ since the distribution can easily be shifted by an arbitrary integer. Finally, in all applications in the cryptographic context that we are aware of one can safely assume that $\sigma \geq 1$.

## 3.1 BLISS Sampler

The BLISS sampler was proposed in [5] as a way to achieve another trade-off between memory requirements and sampling performance for a known distribution. In particular, while plain rejection sampling is very slow, in many settings the memory required to apply any of the generic algorithms described in Section 2 is prohibitively large, especially on constraint devices. The BLISS sampler addresses this issue by only requiring precomputed storage of $O(\log n)$ instead of $O(n)$, while still yielding much better performance than rejection sampling.

The BLISS sampler is a rejection sampler with two main differences to plain rejection sampling. First, it uses a different source distribution, which is still easy to sample, but better adapted to the target distribution, which reduces the rejection rate. Second, a list of logarithmically many precomputed values is used to perform the rejection step through a set of Bernoulli samples. This has the advantage that no FP arithmetic has to be performed during sampling.

The algorithm assumes that $\sigma = k \cdot \sigma_2$ for $\sigma_2 = \sqrt{1/(2\ln 2)}$ and some $k \in \mathbb{Z}$. This is justified since in many settings $\sigma$ is a parameter of the scheme and can be chosen accordingly. The source distribution is defined as $k \cdot \mathcal{D}_{\mathbb{Z},\sigma_2}^{+} + \mathcal{U}(\{0, \ldots, k-1\})$, which can be easily sampled if there is an efficient way to sample $\mathcal{D}_{\mathbb{Z},\sigma_2}^{+}$. By the choice of $\sigma_2$ this can be done using inversion sampling, where the CDT is computed on-the-fly. To see this, note that $\rho_{\sigma_2}(x) = \exp(-\frac{x^2}{2\sigma_2^2}) = \exp(-x^2 \ln 2) = 2^{-x^2}$. So $\sum_{x=0}^{j} \rho(x) = \sum_{x=0}^{j} 2^{-x^2}$ which converges to a constant $\alpha < 2$ as $j \to \infty$. Furthermore, the entries of the CDT correspond exactly to the sums $\sum_{x=0}^{j} 2^{-x^2}$ for $j \in \mathbb{Z}$, which exhibits their binary expansion: the expansion has a one in positions that correspond to squares. Since $(a+1)^2 - a^2 = 2a + 1$, there are exactly $2a$ zeros between the $a$-th one and the $a + 1$-st one for $a < j$. This structure can be exploited by sampling a uniformly random number $r$ in $[0, 2)$ one bit at a time (or in chunks) and checking if $r \in \left[\sum_{x=0}^{j} \rho_{\sigma_2}(x), \sum_{x=0}^{j+1} \rho_{\sigma_2}(x)\right)$ for some $j$. (Note that one can also easily check if $r > \alpha$ and reject $r$ in that case). See [5] for details.

To sample from $\mathcal{D}_{\mathbb{Z},k\sigma_2}^{+}$, a sample from the source distribution is chosen by sampling $x \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_2}^{+}$ and $y \leftarrow \mathcal{U}(\{0, \ldots, k-1\})$, which represent the sample $z = kx + y$. This sample is now accepted with probability $\exp(-\frac{y(y+2kx)}{2\sigma^2})$. Correctness follows from the fact that the probability of outputting a sample $z = kx + y$ is proportional to

$$\rho_{\sigma_2}(x) \exp\left(-\frac{y(y+2kx)}{2\sigma^2}\right) = \exp\left(-\frac{x^2}{2\sigma_2^2} - \frac{y(y+2kx)}{2\sigma^2}\right) = \rho_{\sigma}(z).$$

It remains to show how to perform the rejection step without FP arithmetic and little precomputed memory. For this, recall that an integer sample $z \in \{0, \ldots, \tau\sigma\}$ that is output by the source distribution, is accepted with probability $\exp(-\frac{f(z)}{2\sigma^2})$ for some integer $f(z) \leq z^2$. As seen above, $f(z)$ is easily computable from $z$ using only small integer arithmetic and $\tau$ can be considered a constant. The corresponding Bernoulli trial is realized by storing the numbers $c_i = \exp(-\frac{2^i}{2\sigma^2})$ for all $i \in [\lceil 2\log(\tau\sigma) \rceil]$ in memory. Then the Bernoulli trial is performed by considering the binary representation of $f(z) = \sum_i z_i 2^i$ and rewriting

$$\exp\left(-\frac{f(z)}{2\sigma^2}\right) = \exp\left(-\frac{\sum_i z_i 2^i}{2\sigma^2}\right) = \prod_i \exp\left(-\frac{z_i 2^i}{2\sigma^2}\right) = \prod_{i:z_i=1} \exp\left(-\frac{2^i}{2\sigma^2}\right).$$

This shows that this trial can be carried out by a series of at most $l = \lceil 2\log(\tau\sigma) \rceil$ Bernoulli trials with bias of the form $c_i$ for some $i$. It is clear that this works if storing the numbers $c_i$ exactly, but this is clearly impossible in practice. The next lemma shows that we can approximate the biases $c_i$ and achieve small relative error.

**Lemma 5.** *Let $\mathcal{D}$ be the output distribution of the BLISS sampler with biases $c_i$, and $\widehat{\mathcal{D}}$ be the same algorithm with biases $\widehat{c}_i$. If $\delta_{\mathrm{RE}}(c_i, \widehat{c}_i) \leq \mu$ for all $i$ then $\delta_{\mathrm{RE}}(\mathcal{D}, \widehat{\mathcal{D}}) \leq 2l\mu + O(l^2\mu^2)$.*

*Proof.* Since $\widehat{c}_i \in (1 \pm \mu)c_i$, the rejection probability for any $x$ is

$$\prod_{i:z_i=1} \widehat{c}_i \in \prod_{i:z_i=1} (1 \pm \mu)c_i \in (1 \pm \mu)^l \prod_{i:z_i=1} c_i \in (1 \pm (l\mu + O(\mu^2))) \prod_{i:z_i=1} c_i.$$

The result follows from Lemma 2 (cf. Remark 1). □

Lemma 5 shows that the BLISS sampler preserves the relative error up to a multiplicative loss of $\sim 2l = O(\log\sigma)$ if using FP numbers to approximate the biases $c_i$.

**Remark 3.** *The work of [3] analyzed the required precision for the biases of the BLISS sampler by viewing the cryptosystem as using samples from the respective Bernoulli samplers, rather than from the discrete Gaussian. This requires fixing the number of Bernoulli samples a priori, which is possible in security analyses because one is usually only interested in bounded adversaries and it is reasonable to charge the adversary resources for obtaining Bernoulli samples. In comparison, our result is of a statistical nature and thus slightly stronger: the relative error of the distribution is bounded independently of the number of times the Bernoulli samplers are invoked, which is unbounded in the worst case. Furthermore, our approach has the advantage of being simpler and more modular.*

## 3.2 Karney's Algorithm

Karney's algorithm is a type of rejection algorithm that efficiently samples from the discrete Gaussian. (In fact, the work [13] also shows how to sample from the continuous Gaussian, but we are only interested in discrete distributions here.) We give a description of Karney's algorithm in Alg. 2. Sampling from the non-negative unit discrete Gaussian in Step 2 can be done using only uniform Bernoulli trials. Similarly, the Bernoulli trial in Step 12 is also done by cleverly dividing it up into a series of uniform Bernoulli trials. How exactly these two steps are carried out is not important here.

---
**Algorithm 2** Karney's Algorithm
---
$\textsc{Karney}(\sigma, c)$

| | |
|---|---|
| 1 | **while** true |
| 2 | $\quad t \leftarrow \mathcal{D}_{\mathbb{Z},1}^+$ |
| 3 | $\quad s \leftarrow \{-1, 1\}$ |
| 4 | $\quad j \leftarrow \{0, \ldots, \lceil\sigma\rceil - 1\}$ |
| 5 | $\quad i \leftarrow \lceil t\sigma + sc \rceil$ |
| 6 | $\quad \bar{x} \leftarrow i - (t\sigma + sc)$ |
| 7 | $\quad x \leftarrow \frac{\bar{x}+j}{\sigma}$ |
| 8 | $\quad$ **if** $x \geq 1$ |
| 9 | $\quad\quad$ **continue** // reject |
| 10 | $\quad$ **if** $t = 0$ **and** $x = 0$ **and** $s < 0$ |
| 11 | $\quad\quad$ **continue** // reject |
| 12 | $\quad b \leftarrow \mathcal{B}_p$ where $p = \exp(-\frac{1}{2}x(2t+x))$ |
| 13 | $\quad$ **if** $b = 1$ |
| 14 | $\quad\quad$ **return** $s(i+j)$ |
---

Karney presented his algorithm as an exact algorithm using only integer arithmetic. The algorithm assumes that the parameters are given as rationals. However, a common use case is to apply this sampler on a regular PC using floating point numbers, for example during trapdoor sampling. In this context the parameters are given as floating point numbers, not as rationals. While floating point numbers can be naturally viewed as rationals, doing so in the straight-forward manner leads to relatively large integers. Inspecting the algorithm, it is clear that the integers involved can have roughly twice the size of the mantissa of the floating point numbers of the input parameters. So if working with regular double precision floating point numbers ($k = 53$), the numbers may be larger than 100 bits, which exceeds the length of integers on most architectures (a reasonable target could be 64 bits).

### 3.2.1 A Counterexample

The first natural question is if Algorithm 2 simply preserves a relative error of $\mu = 2^{-k}$ if one replaces the exact integer arithmetic with FP operations. We now give a counterexample showing that this is not true. For concreteness we set $k = 4$ (but note that this example can be generalized to any $k$), $c = 0$ and $\sigma = 1 + \frac{1}{8}$. We consider the case where $t = 7$ in Step 2, $s = 1$ and $j = 1$. In that case we have that $t\sigma = 7 + \frac{7}{8}$, which is not a 4-bit FP number so it needs to be rounded. The closest 4-bit number is 8, so $i = 8$ in Step 5 and $\bar{x} = 0$, while the correct value of $\bar{x}$ would have been $\frac{1}{8}$. This will result in $x = \frac{7}{8}$ (the closest 4-bit FP value to $\frac{1}{\sigma} = \frac{8}{9}$), while it should have been 1. So the check in Step 8 will pass, even though it should have been rejected, and the value $s(i + j) = 9$ will be accepted with some positive probability.

On the other hand, the value 9 can also be output in case $t = 8$ and $j = 0$ with about the same probability. In this case $t\sigma$ is in fact an integer and no rounding errors occur. This shows that 9 has about twice as much relative probability than it should have, which means that the relative error of the distribution is a constant.

### 3.2.2 A Floating Point Version

We are interested in a floating point version of Karney's algorithm. We will assume that the input is given as floating point numbers and that we only have floating point numbers of the same precision available. Furthermore, we assume that we have integers available with a specific bit size. This immediately puts an upper bound on the noise parameter, since we need to be able to represent samples from a discrete Gaussian that have more than negligible probability.

We want to bound the relative error achieved by the modified algorithm. We will show that we can perform all the steps in Karney's algorithm exactly, even with low precision numbers, except step 12 as this requires to compute $x$. Instead, we will approximate $x$, which allows to carry out an approximate Bernoulli trial (in the same way as in Karney's algorithm). The main observation is that Lemma 2 now applies: this approximation of the bias in the last rejection step only leads to a small difference in the relative probability for each potential sample $s(i + j)$ and thus a small relative error.

We will require a little background on floating point numbers, in particular we identify a set of operations on FP numbers with fixed precision that can be carried out exactly without resorting to FP numbers with higher precision.

**Fact 1.** *Given two $k$-bit FP numbers $a$ and $b$ we can easily compute*

- $a \leq b$, $a < b$, $a = b$

- $(a)_{\%1}$, *the reduction* $\mod 1$, *i.e. separating out the fractional part of $a$*

- $2^t a$ *for any $t \in \mathbb{Z}$*

- $\lfloor a \rfloor$ *and* $\lceil a \rceil$

*exactly as $k$-bit FP numbers.*

**Lemma 6** (Sterbenz Lemma). *Given two $k$-bit FP numbers $a$ and $b$ with $\frac{1}{2} \leq \frac{a}{b} \leq 2$, their difference $a - b$ is also a $k$-bit FP number.*

**Corollary 1.** *Given three positive FP numbers $a$, $b$, and $c$, we can check if $(a+b) \circ c$ for any $\circ \in \{\leq, \geq, <, >, =\}$.*

*Proof.* Algorithm 3 performs the check for $\leq$. Its correctness follows from Sterbenz lemma. The same approach works for any other comparison operator. $\square$

---

**Algorithm 3** An exact algorithm to check if $a + b \leq c$ for given FP numbers $a, b, c$

---

$\text{COMPARESUM}_{\text{FP}}(a, b, c)$

 1    **if** $a > c$ or $b > c$
 2       **return** false
 3    **if** $a > \frac{c}{2}$
 4       **return** $b \leq c - a$
 5    **if** $b > \frac{c}{2}$
 6       **return** $a \leq c - b$
 7    **return** true

---

**Lemma 7.** *Given a $k$-bit FP number $a \geq 1$ and an integer $t$, we can compute $(ta)_{\%1}$ and $\lfloor ta \rfloor$ using $\lceil \log t \rceil$ operations, assuming integer data types that are capable of representing $\lfloor ta \rfloor$.*

*Proof.* First note that $(a)_{\%1}$ is a $(k-1)$-bit fixed point number. Using simple peasant multiplication with $k$-bit fixed point arithmetic (e.g. emulated using $k$-bit FP arithmetic with fixed exponent) we can compute $(ta)_{\%1}$ and $\lfloor t(a)_{\%1} \rfloor$. Finally, compute $\lfloor ta \rfloor = t\lfloor a \rfloor + \lfloor t(a)_{\%1} \rfloor$. $\square$

We remark that in the case where the available integer type has bit size larger than the mantissa of the FP numbers plus $\lceil \log t \rceil$, the task in Lemma 7 can be achieved in a much simpler way: multiply $t$ with the mantissa of $a$, which allows to directly read off the two desired values. This case is common on standard CPUs, where the FP numbers have 53 bits of precision and the common integer type has 63 bits. Furthermore, in our setting $\log t < 4$ with overwhelming probability. On the other hand, if working with 80-bit extended precision, where FP numbers commonly have a 63-bit mantissa, this is not the case anymore.

---

**Algorithm 4** Rewrite of Karney's Algorithm

---

$\text{KARNEYREWRITE}(\sigma, c)$

 1    **while** true
 2       $t \leftarrow \mathcal{D}_{\mathbb{Z},1}^{+}$
 3       $s \leftarrow \{-1, 1\}$
 4       $j \leftarrow \{0, \dots, \lceil \sigma \rceil - 1\}$
 5       **if** $\text{CHECKREJECTA}(\sigma, c, t, s, j)$
 6          **continue** // reject
 7       **if** $\text{CHECKREJECTB}(\sigma, c, t, s, j)$
 8          **continue** // reject
 9       $i \leftarrow \text{COMPUTEI}(\sigma, c, t, s)$
10      $x \leftarrow \text{COMPUTEX}(\sigma, c, t, s, j)$
11      $b \leftarrow \mathcal{B}_p$
         where $p = \exp(-\frac{1}{2}x(2t + x))$
12      **if** $b = 1$
13         **return** $s(i + j)$

$\text{COMPUTEI}(\sigma, c, t, s)$

 1    **return** $\lceil t\sigma + sc \rceil$

$\text{COMPUTEX}(\sigma, c, t, s, j)$

 1    $i \leftarrow \text{COMPUTEI}(\sigma, c, t, s)$
 2    **return** $\frac{i - (t\sigma + sc) + j}{\sigma}$

$\text{CHECKREJECTA}(\sigma, c, t, s, j)$

 1    $x \leftarrow \text{COMPUTEX}(\sigma, c, t, s, j)$
 2    **return** $x \geq 1$

$\text{CHECKREJECTB}(\sigma, c, t, s, j)$

 1    $x \leftarrow \text{COMPUTEX}(\sigma, c, t, s, j)$
 2    **return** ($t = 0$ and $x = 0$ and $s < 0$)

---

We begin our analysis by rewriting Algorithm 2 using auxiliary procedures that encapsulate the critical steps (see Algorithm 4). Then we will present alternative procedures that can be implemented with FP numbers such that the entire procedure is guaranteed to yield a small relative error in the output distribution.

In the following, we show how to implement COMPUTEI, CHECKREJECTA and CHECKREJECTB exactly using FP numbers. Finally, we will introduce a procedure that approximates $x$ such that the computed bias $p$ has small relative error. Then the result follows from Lemma 2.

**Lemma 8.** *For any set of inputs* $(t, s, \sigma, c) \in \mathbb{Z}_+ \times \{-1, 1\} \times \mathbb{R}_+ \times [0, 1)$

$$\text{COMPUTEI}_{\text{FP}}(\sigma, c, t, s) = \text{COMPUTEI}(\sigma, c, t, s)$$

*where* $\sigma$ *and* $c$ *are given as FP numbers and* $\text{COMPUTEIFP}(t, s, \sigma, c)$ *is implemented using FP numbers of the same precision.*

*Proof.* Let $a = \lfloor t\sigma \rfloor$ and $b = (t\sigma)_{\%1}$. Then $\lceil t\sigma + sc \rceil = a + \lceil b + sc \rceil$. Since $c \in [0, 1)$, there are only 3 possible values: $\lceil t\sigma + sc \rceil \in \{a, a+1, a+2\}$. We can split this up into the cases where $s = -1$ and $s = 1$. In the former case, we have

$$\lceil t\sigma + sc \rceil = \begin{cases} a & \text{if } b \leq c \\ a+1 & \text{otherwise} \end{cases}$$

In the case $s = 1$ we have three cases:

$$\lceil t\sigma + sc \rceil = \begin{cases} a & \text{if } b = c = 0 \\ a+1 & \text{if } b + c \leq 1 \\ a+2 & \text{otherwise} \end{cases}$$

Recall that the second case can be checked using Corollary 1. $\square$

---

**Algorithm 5** An exact algorithm to compute $i = \lceil t\sigma + sc \rceil$ using FP numbers, where $t \in \mathbb{Z}$, $s \in \{-1, 1\}$ and $c \in [0, 1)$

---

$\text{COMPUTEI}_{\text{FP}}(\sigma, c, t, s)$

1     $i \leftarrow \lfloor t\sigma \rfloor$
2     $b \leftarrow (t\sigma)_{\%1}$
3     **if** $s < 0$ and $b > c$
4        $i \leftarrow i + 1$
5     **if** $s > 0$ and ($b > 0$ or $c > 0$)
6        **if** $b + c \leq 1$      // cf. Corollary 1
7           $i \leftarrow i + 1$
8        **else**
9           $i \leftarrow i + 2$
10    **return** $i$

---

**Lemma 9.** *For any set of inputs* $(t, s, \sigma, c, j) \in \mathbb{Z}_+ \times \{-1, 1\} \times \mathbb{R}_+ \times [0, 1) \times \mathbb{Z}_+$

$$\text{CHECKREJECTA}_{\text{FP}}(\sigma, c, t, s, j) = \text{CHECKREJECTA}(\sigma, c, t, s, j)$$

*where* $\sigma$ *and* $c$ *are given as FP numbers and* $\text{CHECKREJECTA}_{\text{FP}}$ *is implemented using FP numbers of the same precision.*

*Proof.* As in the original algorithm, denote with $\bar{x} = i - \lceil t\sigma + sc \rceil$. Then this check is equivalent to $\bar{x} + j \geq \sigma$. If $\bar{x} = 0$ this must always be false by the choice of $j$ so by correctness of ISXBARZERO (see below) we can assume $\bar{x} \in (0, 1)$. Furthermore, the check can only be true if $j = \lfloor \sigma \rfloor$, because $c \in [0, 1)$. In this case this is equivalent to $\bar{x} \geq (\sigma)_{\%1}$. Note that $\bar{x} = 1 - (t\sigma + sc)_{\%1}$, so this is the same as

$$1 \geq (t\sigma + sc)_{\%1} + (\sigma)_{\%1}. \tag{1}$$

We consider the case $s > 0$. Then we have

$$(t\sigma + c)_{\%1} = \begin{cases} (t\sigma)_{\%1} + c - 1 & \text{if } (t\sigma)_{\%1} + c \geq 1 \\ (t\sigma)_{\%1} + c & \text{else} \end{cases}$$

Plugging this into (1)

$$1 \geq (t\sigma + sc)_{\%1} + (\sigma)_{\%1} = \begin{cases} (t\sigma)_{\%1} + (\sigma)_{\%1} + c - 1 & \text{if } (t\sigma)_{\%1} + c \geq 1 \\ (t\sigma)_{\%1} + (\sigma)_{\%1} + c & \text{else} \end{cases}$$

Note that we can distinguish the cases using Corollary 1. In each case we can compute $(t\sigma)_{\%1} + (\sigma)_{\%1}$ exactly and invoke Corollary 1, with the left-hand side being either 2 or 1, respectively.

Next consider the case $s < 0$. Then (1) is equivalent to $1 \geq (t\sigma - c)_{\%1} + (\sigma)_{\%1}$. Note that

$$(t\sigma - c)_{\%1} = \begin{cases} (t\sigma)_{\%1} - c & \text{if } (t\sigma)_{\%1} \geq c \\ (t\sigma)_{\%1} - c + 1 & \text{else.} \end{cases}$$

Again we can rewrite

$$1 \geq (t\sigma - c)_{\%1} + (\sigma)_{\%1} = \begin{cases} (t\sigma)_{\%1} + (\sigma)_{\%1} - c & \text{if } (t\sigma)_{\%1} \geq c \\ (t\sigma)_{\%1} + (\sigma)_{\%1} - c + 1 & \text{else.} \end{cases}$$

Again we can compute $(t\sigma)_{\%1} + (\sigma)_{\%1}$ exactly. Note that in the latter case the 1 cancels and we can directly check the inequality. In the former case, we check if $(t\sigma)_{\%1} + (\sigma)_{\%1} < 1$ in which case the inequality is obviously true. Otherwise, we subtract 1 from both sides which yields $0 \geq ((t+1)\sigma)_{\%1} - c$, which we can obviously check with $c \geq ((t+1)\sigma)_{\%1}$. □

It remains to prove correctness of IsXbarZero$_\text{FP}$.

**Lemma 10.** *Algorithm 7 returns* **true** *iff* $t\sigma + sc$ *is integral.*

*Proof.* The value $t\sigma + sc$ is integral iff $(t\sigma)_{\%1} + (sc)_{\%1} \in \{0, 1\}$. If $s < 0$, this simply reduces to $(t\sigma)_{\%1} = c$. Otherwise we need to check if $(t\sigma)_{\%1} + c = 1$, which we can do by Corollary 1, or if $t\sigma = c = 0$. □

Performing CheckRejectB exactly is straightforward, since $x = 0$ iff $j = 0$ and $\bar{x} = 0$. Conditioned on $t = 0$, the latter is the case iff $c = 0$ (cf. Algorithm 8).

The above shows that Algorithm 4 behaves identical if we replace ComputeI, CheckRejectA and CheckRejectB with ComputeI$_\text{FP}$, CheckRejectA$_\text{FP}$ and CheckRejectB$_\text{FP}$, respectively. Clearly, we cannot hope to implement ComputeX exactly, since the result might not even be a $k$-bit FP number (where $k$ is the precision of the parameters). However, it is sufficient to approximate $x$ well enough in order to maintain a small relative error in the output distribution. We begin with the observation that it is sufficient to approximate $x$ with a small *absolute or relative* error $\mu$. Standard calculations show that in either case the term $\exp(-\frac{1}{2}\hat{x}(2t + \hat{x}))$ will be off by a multiplicative factor $\sim \exp(t\mu) \approx 1 + t\mu$, which shows that the relative error is only about $t\mu$.

**Algorithm 6** An exact algorithm to check if $x \geq 1$

CHECKREJECTA$_{\text{FP}}(\sigma, c, t, s, j)$

```
 1    if j < ⌊σ⌋ or IsXBARZERO_FP(σ, c, t, s)
 2       return false
 3    b ← (tσ)_{%1}
 4    a ← (σ)_{%1}
 5    z ← a + b
 6    if s > 0
 7       if b + c ≥ 1
 8          return (z + c ≤ 2)
 9       else
10          return (z + c ≤ 1)
11    else
12       if b > c
13          if z ≤ 1
14             return true
15          else
16             return (c ≥ (z)_{%1})
17       else
18          return (c ≥ z)
```

**Algorithm 7** An exact algorithm to check if $t\sigma + sc$ is integral

IsXBARZERO$_{\text{FP}}(\sigma, c, t, s)$

```
 1    b ← (tσ)_{%1}
 2    if s > 0
 3       if [(b = 0 and c = 0) or (b + c = 1)]
 4          return true
 5    if s < 0 and b = c
 6          return true
 7    return false
```

**Algorithm 8** An exact algorithm to perform CHECK-REJECTB using FP numbers

CHECKREJECTB$_{\text{FP}}(\sigma, c, t, s, j)$

```
 1    return (c = 0 and t = 0 and j = 0 and s < 0)
```

The problematic part is computing $\bar{x} = i - (t\sigma + sc)$ with low error. Recall that we can check if $\bar{x} = 0$, so assume that $\bar{x} \in (0, 1)$, which means $\bar{x} = 1 - (t\sigma + sc)_{\%1}$. While we can compute $b = (t\sigma)_{\%1}$ (cf. Lemma 7) and $sc$ exactly, the sum $a = b + sc$ is computed up to an approximation. Naively extracting the fractional part of $a$ could cause catastrophic errors if $(b + sc)_{\%1}$ is very close to 1 and $a$ is rounded to an integer. In that case, $(a)_{\%1} = 0$, which is very far from $(b + sc)_{\%1}$. This would result in a bad approximation of $\bar{x}$. To avoid this problem, we forgo the $(\cdot)_{\%1}$ computation and check explicitly if $b + sc < 0$ or $b + sc \geq 1$ and adjust $\bar{x}$ accordingly through appropriate additions and subtractions.

To prove a good approximation for Algorithm 9, we need a technical fact that can be easily verified using standard calculations.

**Fact 2.** *Let $\hat{a}$ and $b$ be $k$-bit FP numbers such that $|a - \hat{a}| \leq \mu$. Then*

$$\left| \widehat{\hat{a} + b} - (a + b) \right| \leq \mu + 2^{-k}(a + b) + 2^{-k}\mu \lesssim \mu + 2^{-k}(a + b).$$

**Lemma 11.** *For any set of inputs $\sigma, c, t, s, j \in \mathbb{R}_+ \times [0, 1) \times \mathbb{Z}_+ \times \{-1, 1\} \times \mathbb{Z}_+$ given as $k$-bit FP numbers, if Algorithm 9 is implemented with $k$-bit FP numbers, then its output value $x'$ will satisfy at least one of*

- $|x' - x| = O(2^{-k})$

- $\delta_{\text{RE}}(x', x) = O(2^{-k})$.

*Proof.* Multiple applications of Lemma 2 show that $\bar{x}$, as computed in line 4 to 8 of Algorithm 9, has low absolute error of $O(2^{-k})$, since all the numbers involved have small absolute value. We now make a case distinction: first consider the case $j = 0$. In that case, $x = \frac{\bar{x}}{\sigma}$ will have also small absolute error of $O(2^{-k}/\sigma)$. On the other hand, if $j \geq 1$, $\bar{x} + j$ will actually have relative error $O(2^{-k}/(x + j)) = O(2^{-k})$. Division by $\sigma$ can increase this by at most a factor 2, which shows that also $x$ has relative error of $O(2^{-k})$. $\square$

We are now ready to prove the main theorem of this section.

**Theorem 2.** *If Algorithm 4 is implemented using* CHECKREJECTA$_{\text{FP}}$, COMPUTEI$_{\text{FP}}$, CHECKREJECTB$_{\text{FP}}$ *and* COMPUTEX$_{\text{FP}}$ *with $k$-bit numbers and the exponentiation in step 11 is computed with low relative error, the relative error between its output distribution and the one of Karney's algorithm is $O(2^{-k})$.*

*Proof.* Recall that Algorithm 4 behaves identical to its FP variant up to step 9. Denote the distribution of $(s, i, j)$ that are not rejected up to that point as *source distribution*. The probability that such a sample is rejected has relative error $O(2^{-k})$ compared to the exact variant by Lemma 11. We can now apply Lemma 2 (cf. Remark 1) to finish the proof. $\qquad\square$

We note that the rejection step with value $\exp(-\frac{1}{2}x(2t + x))$ can be either carried out exactly as in Karney's original algorithm or using a fast procedure to approximate $\exp(\cdot)$ with small relative error, if it is available.

---

**Algorithm 9** An algorithm to approximately compute $x$

$\textsc{ComputeX}_{\text{FP}}(\sigma, c, t, s, j)$

1    **if** $\textsc{IsXbarZero}_{\text{FP}}(\sigma, c, t, s)$
2        **return** $\frac{j}{\sigma}$
3    $b \leftarrow (t\sigma)_{\%1}$
4    $\bar{x} \leftarrow 1 - (b + sc)$
5    **if** $s < 0$ and $b < c$
6        $\bar{x} \leftarrow \bar{x} - 1$
7    **if** $s > 0$ and $b + c \geq 1$
8        $\bar{x} \leftarrow \bar{x} + 1$
9    **return** $\frac{\bar{x} + j}{\sigma}$

---

## 3.3   Experimental Results

We implemented our FP version of Karney's algorithm (which we denote by $\textsc{Karney}_{\text{FP}}$ in this section) using standard IEEE double precision floating point numbers and compared it to a public implementation of Karney's algorithm [2], where we set the precision to 100 bits, such that the two algorithms offer similar security guarantees when used in cryptographic primitives. Our implementation uses the same source of randomness as [2], is also written in C and compiled with the same parameters. During our experimentation on a standard PC we found that $\textsc{Karney}_{\text{FP}}$ is about 20 times faster than the high precision variant in [2] on typical parameters. We stress though that this is highly platform dependent and caution that this comparison is merely an indication that the relative performance of $\textsc{Karney}_{\text{FP}}$ can be very good.

We note that [2] also provides an implementation using double precision FP numbers. While this version cannot guarantee a low relative error , as outlined in Section 3.2.1,and thus guarantees a much lower level of security (if any), we point out that our implementation was only roughly 25% slower than the low precision version in [2]. This indicates that on standard PC's the penalty for the additional conditionals required to guarantee low relative error is very acceptable. The source code of our implementation will be made public with the publication of this work.

## References

[1] C. Aguilar-Melchor, M. R. Albrecht, and T. Ricosset. Sampling from arbitrary centered discrete gaussians for lattice-based cryptography. In D. Gollmann, A. Miyaji, and H. Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 3–19. Springer, Heidelberg, July 2017.

[2] M. R. Albrecht and M. Walter. dgs, Discrete Gaussians over the Integers. Available at `https://bitbucket.org/malb/dgs`, 2018.

[3] 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 T. Iwata and J. H. Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 3–24. Springer, Heidelberg, Nov. / Dec. 2015.

[4] D. B. Cousins, G. Di Crescenzo, K. D. Gür, K. King, Y. Polyakov, K. Rohloff, G. W. Ryan, and E. Savas. Implementing conjunction obfuscation under entropic ring LWE. In *2018 IEEE Symposium on Security and Privacy*, pages 354–371. IEEE Computer Society Press, May 2018.

[5] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal Gaussians. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, Heidelberg, Aug. 2013.

[6] L. Ducas and P. Q. Nguyen. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In X. Wang and K. Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 415–432. Springer, Heidelberg, Dec. 2012.

[7] N. C. Dwarakanath and S. D. Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Appl. Algebra Eng., Commun. Comput.*, 25(3):159–180, June 2014.

[8] N. Genise and D. Micciancio. Faster gaussian sampling for trapdoor lattices with arbitrary modulus. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 174–203. Springer, Heidelberg, Apr. / May 2018.

[9] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM Press, May / June 2009.

[10] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In R. E. Ladner and C. Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206. ACM Press, May 2008.

[11] K. D. Gür, Y. Polyakov, K. Rohloff, G. W. Ryan, and E. Savas. Implementation and evaluation of improved gaussian sampling for lattice trapdoors. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '18, pages 61–71, New York, NY, USA, 2018. ACM.

[12] R. A. Hallman, K. Laine, W. Dai, N. Gama, A. J. Malozemoff, Y. Polyakov, and S. Carpov. Building applications with homomorphic encryption. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 18: 25th Conference on Computer and Communications Security*, pages 2160–2162. ACM Press, Oct. 2018.

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

[14] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, Heidelberg, Apr. 2012.

[15] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th Annual Symposium on Foundations of Computer Science*, pages 372–381. IEEE Computer Society Press, Oct. 2004.

[16] D. Micciancio and M. Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 455–485. Springer, Heidelberg, Aug. 2017.

[17] D. Micciancio and M. Walter. On the bit security of cryptographic primitives. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 3–28. Springer, Heidelberg, Apr. / May 2018.

[18] C. Peikert. An efficient and parallel Gaussian sampler for lattices. In T. Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97. Springer, Heidelberg, Aug. 2010.

[19] T. Pöppelmann, L. Ducas, and T. Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In L. Batina and M. Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 353–370. Springer, Heidelberg, Sept. 2014.

[20] T. Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 347–374. Springer, Heidelberg, Dec. 2017.

[21] S. S. Roy, F. Vercauteren, and I. Verbauwhede. High precision discrete Gaussian sampling on FPGAs. In T. Lange, K. Lauter, and P. Lisonek, editors, *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 383–401. Springer, Heidelberg, Aug. 2014.

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

[23] R. K. Zhao, R. Steinfeld, and A. Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. Cryptology ePrint Archive, Report 2018/1234, 2018. `https://eprint.iacr.org/2018/1234`.

[24] Z. Zheng, X. Wang, G. Xu, and C. Zhao. Error estimation of practical convolution discrete gaussian sampling with rejection sampling. Cryptology ePrint Archive, Report 2018/309, 2018. `https://eprint.iacr.org/2018/309`.