SAMPLING FROM DISCRETE GAUSSIANS FOR LATTICE-BASED CRYPTOGRAPHY ON A CONSTRAINED DEVICE

NAGARJUN C. DWARAKANATH AND STEVEN D. GALBRAITH

ABSTRACT. Modern lattice-based public-key cryptosystems require sampling from discrete Gaussian (normal) distributions. The paper surveys algorithms to implement such sampling efficiently, with particular focus on the case of constrained devices with small on-board storage and without access to large numbers of external random bits. We review lattice-based encryption schemes and signature schemes and their requirements for sampling from discrete Gaussians. Finally, we make some remarks on challenges and potential solutions for practical lattice-based cryptography.

Keywords: Lattice-based cryptography, sampling discrete gaussian distributions.

1. INTRODUCTION

Lattice-based cryptography is an extremely active research area that promises systems whose security depends on worst-case computational assumptions in lattices that are believed to be hard even for quantum computers. It is also often said that lattice-based cryptosystems are fast and simple to implement, as the basic operation is only matrix-vector multiplication modulo an integer q. Hence, one might think that lattice-based cryptosystems are ideal for constrained devices (by a constrained device we think of an embedded device or portable device that has small RAM (e.g., measured in kilobytes rather than megabytes), a modest processor, and is required to be economical with respect to power usage). However, there are several reasons why implementing these cryptosystems may not be practical on constrained devices:

(1) The public keys and ciphertexts/signatures may be very large.
(2) The systems often require sampling from discrete Gaussian distributions on lattices.

It is the second of these issues that is the focus of our paper. The problem is to sample a vector from a discrete Gaussian distribution on a lattice $L \subseteq \mathbb{Z}^m$. This task can be reduced to sampling m integers from discrete Gaussian distributions on $\mathbb{Z}$. While sampling from Gaussian distributions is a standard problem in statistical computing, there are several reasons why implementing it can be inconvenient on a constrained device:

(1) The methods require the generation of a large number of random bits as input to the sampling algorithm. Generating these bits (e.g., using a pseudorandom generator) may have non-negligible cost.
(2) The sampling algorithms require either high-precision floating-point arithmetic or very large pre-computed tables.
(3) Cryptographical applications require higher quality sampling (i.e., a smaller statistical difference between the desired distribution and the actual distribution sampled) than traditional applications in statistical computing.

Hence, it is not clear whether it is practical to sample from discrete Gaussian distributions on a constrained device. The aims of this paper are:

- To discuss some issues that have not been widely addressed in the literature on lattice cryptography.

- To determine the practicality of currently proposed schemes with rigorous security guarantees, with direct reference to the proposed algorithms and suggested parameters.
- To survey some methods for compact and efficient sampling from discrete Gaussians, and give an analysis of them.

The paper is organised as follows. Section 2 recalls some basic definitions and results about discrete Gaussian distributions on lattices. Section 3.1 recalls some encryption schemes based on lattices and discusses which schemes require sampling from discrete Gaussians. Section 3.2 recalls some provably secure lattice-based signature schemes. Some of these are based on trapdoor lattices and the "hash-and-sign" paradigm, while others are based on the Fiat-Shamir paradigm. Our finding is that there is no hope to implement hash-and-sign signatures on a constrained device.

Section 4.1 recalls the standard methods to compute numerical approximations to the function $\exp(x)$. Finally, in Section 5 we recall the Knuth-Yao algorithm and analyse its storage cost. Section 6 suggests an alternative approach to sampling from discrete Gaussian distributions by using the central limit theorem, and explains why this does not seem to be useful in practice. Section 7 mentions some concurrent and subsequent papers on this topic.

2. Discrete Gaussian Distributions

Let $\sigma \in \mathbb{R}_{>0}$ be fixed. The *discrete normal distribution* or *discrete Gaussian distribution* on $\mathbb{Z}$ with mean 0 and parameter σ (note that σ is close, but not equal, to the standard deviation of the distribution) is denoted D_σ and is defined as follows. Let

$$S = \sum_{k=-\infty}^{\infty} e^{-k^2/(2\sigma^2)} = 1 + 2\sum_{k=1}^{\infty} e^{-k^2/(2\sigma^2)} \tag{1}$$

and let E be the random variable on $\mathbb{Z}$ such that, for $x \in \mathbb{Z}$, $\Pr(E = x) = \rho_\sigma(x) = \frac{1}{S}e^{-x^2/(2\sigma^2)}$. Note that some authors write the probability as proportional to $e^{-\pi x^2/s^2}$, where $s = \sqrt{2\pi}\sigma$ is a parameter.

More generally we will use the following notation. For $\sigma, c \in \mathbb{R}$ we define $\rho_{\sigma,c}(x) = \exp(-(x - c)^2/(2\sigma^2))$. We write

$$S_{\sigma,c} = \rho_{\sigma,c}(\mathbb{Z}) = \sum_{k=-\infty}^{\infty} \rho_{\sigma,c}(k)$$

and define $D_{\sigma,c}$ to be the distribution on $\mathbb{Z}$ such that the probability of $x \in \mathbb{Z}$ is $\rho_{\sigma,c}(x)/S_{\sigma,c}$.

One can extend these definitions to a lattice $L \subseteq \mathbb{R}^m$. For $\mathbf{x} \in L$ and $\mathbf{c} \in \mathbb{R}^m$ we have $\rho_{L,\sigma,\mathbf{c}}(\mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{c}\|^2/(2\sigma^2))$ and $D_{L,\sigma,\mathbf{c}}$ is the distribution on L given by $\Pr(\mathbf{x}) = \rho_{L,\sigma,\mathbf{c}}(\mathbf{x})/\rho_{L,\sigma,\mathbf{c}}(L)$. We write $D_{L,\sigma}$ for $D_{L,\sigma,0}$.

We now mention some tail bounds for discrete Gaussians. Lemma 4.4(1) of the full version of [18] states that

$$\Pr_{z \leftarrow D_\sigma}(|z| > 12\sigma) \leq 2\exp(-12^2/2) < 2^{-100}. \tag{2}$$

For the lattice distribution, Lemma 4.4(3) of the full version of [18] states that if $\mathbf{v}$ is sampled from $D_{\mathbb{Z}^m,\sigma}$ then

$$\Pr_{z \leftarrow D_{\mathbb{Z}^m,\sigma}}(\|\mathbf{v}\| > c\sigma\sqrt{m}) < c^m \exp(m(1 - c^2)/2). \tag{3}$$

Lemma 1. *Let $\sigma > 0$ and $m \in \mathbb{N}$ be fixed. Consider the distribution $D_{\mathbb{Z}^m,\sigma}$. Let $k \in \mathbb{N}$ and suppose $c \geq 1$ is such that*

$$c > \sqrt{1 + 2\log(c) + 2(k/m)\log(2)}.$$

Then

$$\Pr_{\boldsymbol{v} \leftarrow D_{\mathbb{Z}^m,\sigma}}(\|\boldsymbol{v}\| > c\sqrt{m}\sigma) < \frac{1}{2^k}.$$

In particular, if $k = 100$ *and* $m = 512$ *then one can take* $c = 1.388$ *and* $c\sqrt{m}\sigma = 31.4\sigma$. *Similarly, if* $k = 100$ *and* $m = 1024$ *then one can take* $c = 1.275$ *and* $c\sqrt{m}\sigma = 40.8\sigma$.

Proof. The result follows immediately from equation (3). Solving $c^m \exp(m(1-c^2)/2) < 1/2^k$ gives the result. □

2.1. **Sampling Methods.** There are standard methods in statistical computing for sampling from such distributions. The book by Devroye [5] gives a thorough survey. Two basic methods are rejection sampling and the inversion method. Rejection sampling (Section II.3 of [5]) from a set S involves sampling $x \in S$ from some easy distribution (typically the uniform or exponential distribution) and then accepting the sample with probability proportional to $\Pr(x)$ (we give more details below). The inversion method (Section II.2 of [5]) is to use a function or table that translates sampling from the required distribution into sampling from a uniform distribution on a different set. The formulation in [5] is to have a continuous distribution function $F : S \to [0,1]$ such that if U is a uniform random variable on $[0,1]$ then $F^{-1}(U)$ is a random variable on S of the required form. When S is a finite set this can be implemented as a table of values $F(x) \in [0,1]$ over $x \in S$, with $\Pr(F(x) \le u) = \Pr(x \le F^{-1}(u))$.

A standard assumption in works such as [5] is that one can sample uniformly from $[0,1]$. This is achieved in practice by sampling a sequence of uniformly chosen bits, giving the binary expansion of a real number between 0 and 1.

It is useful to have a theoretical lower bound on the number of uniformly chosen bits required, on average, to sample from a given distribution. The inversion method gives an algorithm to sample from the distribution that attains this theoretical minimum. The lower bound is given by the entropy of the distribution, which is a measure of the "information" to specify a value in the distribution. The entropy of the normal distribution on $\mathbb{R}$ with variance σ^2 is $\log(2\pi e\sigma^2)/2 \approx 1.4 + \log(\sigma)$. Since we measure bitlengths we prefer to use $\log_2$. In the discrete case the definition of entropy is

$$H = -\sum_{k=-\infty}^{\infty} p_k \log_2(p_k)$$

where $p_k = \rho_\sigma(k)$. For example, we calculated the entropy of the discrete Gaussian distribution on $\mathbb{Z}$ with parameter $\sigma = 20$ to be $H \approx 6.36$. Increasing σ to 200 gives $H \approx 9.69$. In other words, one should only need to use around 6 uniformly generated bits to generate a single sample from D_{20}, and around 10 bits to generate a sample from D_{200}. More generally, one should only need around $2+\log_2(\sigma)$ uniformly generated bits to sample an integer from D_σ. Hence, we should be able to have efficient algorithms to sample from these distributions that only use a small number of calls to a (pseudo-)random bit generator.

We now give more details about rejection sampling (Section II.3 of [5]). Let S be a finite set to be sampled, let f be the desired probability distribution on S and g the probability distribution that can be easily sampled (e.g., $g(x) = 1/\#S$ for all $x \in S$). Let $M_R \in \mathbb{R}$ be such that $M_R g(x) \ge f(x)$ for all $x \in S$. The sampling algorithm is to sample $x \in S$ according to the distribution g and then to output x with probability $\frac{f(x)}{M_R g(x)}$ and otherwise to repeat the process. The expected number of times x is sampled before an output is produced is M_R. In the case where g is the uniform distribution on the set S, the rejection sampling algorithm requires on average $M_R \log_2(\#S)$ calls to the random bit generator.

2.2. **Statistical distance.** No finite computation will ever produce a discrete normal distribution. Since we want efficient cryptosystems we need algorithms with bounded running time that sample from distributions that are very close to the desired distribution. The precision required is governed by the security proof of the cryptosystem. To make this precise one uses the notion of statistical difference. Let X and Y be two random variables corresponding to given distributions on a lattice L. Then the statistical difference is defined by

$$\Delta(X,Y) = \tfrac{1}{2}\sum_{\mathbf{x}\in L}\Big|\Pr(X=\mathbf{x}) - \Pr(Y=\mathbf{x})\Big|.$$

It is necessary for the rigorous security analysis that the statistical difference between the actual distribution being sampled and the theoretical distribution (as used in the security proof) is negligible, say around 2^{-90} to 2^{-128}.

We now prove a simple fact about how well one needs to approximate the discrete Gaussian on $\mathbb{Z}$ in order to generate good samples from the discrete Gaussian on $\mathbb{Z}^m$.

Lemma 2. *Let $\sigma > 0, \epsilon > 0$ be given. Let $k \in \mathbb{N}$ and let $t > 0$ be such that the tail bound $\Pr(\|\mathbf{v}\| > t\sigma)$ as in Lemma 1 is at most $1/2^k$. For $x \in \mathbb{Z}$ denote by ρ_x the probability of sampling x from the distribution D_σ. Suppose one has computed approximations $0 \le p_x \in \mathbb{Q}$ for $x \in \mathbb{Z}$, $-t\sigma \le x \le t\sigma$ such that*

$$|p_x - \rho_x| < \epsilon$$

and such that $\sum_{x=-t\sigma}^{t\sigma} p_x = 1$. Let D' be the distribution on $[-t\sigma, t\sigma] \cap \mathbb{Z}$ corresponding to the probabilities p_x. Denote by D'' the distribution on $\mathbb{Z}^m$ corresponding to taking m independent samples v_i from D' and forming the vector $\mathbf{v} = (v_1, \dots, v_m)$. Then

$$\Delta(D'', D_{\mathbb{Z}^m,\sigma,0}) < 2^{-k} + 2mt\sigma\epsilon. \tag{4}$$

Proof. Let $X = \{\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{Z}^n : |v_i| \le t\sigma\}$ be the support of D''. By restricting to sampling entries of $\mathbf{v}$ from $[-t\sigma, t\sigma] \cap \mathbb{Z}$ we certainly cover all vectors $\mathbf{v}$ such that $\|\mathbf{v}\| \le t\sigma$ (and many more). Hence, by assumption, the sum of all the probabilities with respect to $D_{\mathbb{Z}^m,\sigma}$ for all $\mathbf{v} \in \mathbb{Z}^m - X$ is less than 2^{-k}.

It remains to compute the statistical difference on X. In other words, to bound

$$\sum_{\mathbf{v}\in X} |p_{v_1}p_{v_2}\cdots p_{v_m} - \rho_{v_1}\cdots\rho_{v_m}|. \tag{5}$$

One can write the inner term $p_{v_1}\cdots p_{v_m} - \rho_{v_1}\cdots\rho_{v_m}$ as

$$\begin{aligned} & p_{v_1}p_{v_2}\cdots p_{v_m} - \rho_{v_1}p_{v_2}\cdots p_{v_m} + \rho_{v_1}p_{v_2}\cdots p_{v_m} - \rho_{v_1}\cdots\rho_{v_m} \\ = \ & (p_{v_1} - \rho_{v_1})(p_{v_2}\cdots p_{v_m}) + \rho_{v_1}(p_{v_2}\cdots p_{v_m} - \rho_{v_2}\cdots\rho_{v_m}). \end{aligned}$$

Continuing the process gives

$$(p_{v_1} - \rho_{v_1})(p_{v_2}\cdots p_{v_m}) + \rho_{v_1}(p_{v_2} - \rho_{v_2})(p_{v_3}\cdots p_{v_m}) + \cdots + (\rho_{v_1}\cdots\rho_{v_{m-1}})(p_{v_m} - \rho_{v_m}).$$

Applying the triangle inequality turns the sum into a sum of entries of the form (this is just the first one)

$$|p_{v_1} - \rho_{v_1}| \sum_{\mathbf{w}\in X\cap\mathbb{Z}^{m-1}} p_{w_2}\cdots p_{w_m}$$

where $X \cap \mathbb{Z}^{m-1}$ means restricting to the last $m-1$ components. Since the probabilities sum to 1, each term is bounded above by $|p_{v_1} - \rho_{v_1}|$. Since there are $2t\sigma$ choices for each v_i and m choices for i it follows that equation (5) is bounded by $2mt\sigma\epsilon$. □

The above result is not tight as $[-t\sigma, t\sigma]^m$ is a hypercube while $\{\mathbf{v} \in \mathbb{Z}^m : \|\mathbf{v}\| > t\sigma\}$ is the exterior of a hypersphere, so we are double counting. Hence, the dependence on m may not actually be as bad in practice as the linear term in equation (4) would suggest. We performed some toy experiments to assess this. We fixed small values for σ, t, ϵ (e.g., $(\sigma, t, \epsilon) = (3, 4, 0.01)$) and computed the statistical distance of the distributions $D_{\mathbb{Z}^m,\sigma}$ and D'', as above, for $m = 1, 2, \dots$. For ease of computation we restricted to the

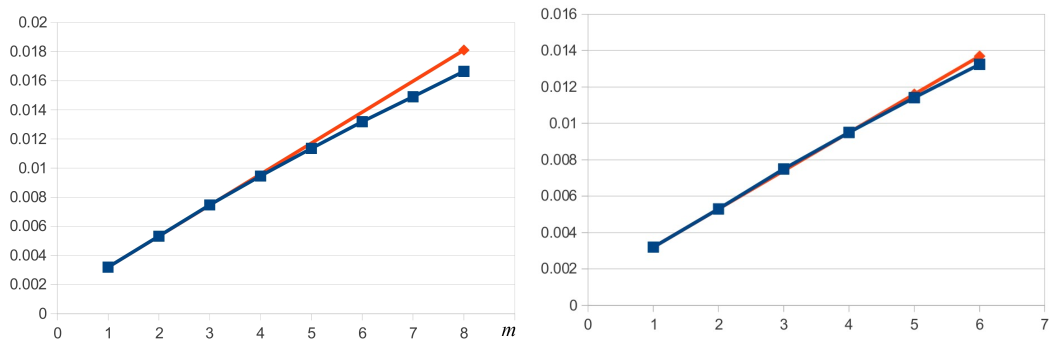

FIGURE 1. Plot of statistical distance Δ (for one orthont of $\mathbb{Z}^m$) versus m for $\sigma = 3$ and $\epsilon = 0.01$ (blue line, square dots). The straight line (red, diamonds) is a linear interpolation of the data points for $m = 1$ and $m = 2$. The left hand graph has $t = 4$ and the right hand graph has $t = 6$.

first orthont $\mathbb{Z}^m_{\geq 0}$. A typical graph is given in Figure 1. One sees that the graph drops below linear extremely slowly, however this seems to be mainly due to the tail in the region of the hypercube that is not in the hypersphere. As t grows (making the tail smaller) the plot becomes closer to linear. The results indicate that there is still a linear dependence on m. So for the remainder of the paper we assume that it is necessary to have the bound growing linearly with m.

Note that m hurts us twice. First, there is a $\sqrt{m}$ contribution to t from Lemma 1. Second, there is the m in the term $2mt\sigma\epsilon$ term in equation (4). It therefore follows that if we want statistical distance for the lattice sampling of around 2^{-k}, then we need to sample from $\mathbb{Z}$ with statistical distance around $2^{-k}/(2m\sqrt{m}\sigma)$.

Taking $k = 100, m = 1024, t = 40.8$ and $\sigma = 1.6 \cdot 10^5$ in Lemma 2 gives statistical distance of approximately $2^{-100} + 1.3 \cdot 10^{10}\epsilon$. Hence, to have statistical difference of 2^{-90} it is necessary to take $\epsilon \approx 2^{-124}$.

3. LATTICE-BASED CRYPTOGRAPHY

Lattice-based cryptography is a large subject and there are a number of different cryptosystems with a number of special features. The aim of this section is to recall some examples of these systems, explain how discrete Gaussian distributions are used, and to discuss some specific cryptosystems that one may wish to use on constrained devices.

3.1. **Learning with errors and encryption schemes.**

As a first example we recall the learning with errors problem (LWE) and the basic LWE encryption scheme, invented by Regev [23, 24].

Let $q, n \in \mathbb{N}$, where q is an odd integer. We represent $\mathbb{Z}/q\mathbb{Z}$ using the set $\{-(q-1)/2, \ldots, (q-1)/2\}$. We represent the set $(\mathbb{Z}/q\mathbb{Z})^n$ using column vectors of length n with entries in $\mathbb{Z}/q\mathbb{Z}$. The entries of the column vector $\mathbf{s}$ will be denoted $s_1, \ldots, s_n$.

Definition 1. Let n, q, σ be as above. Let $\mathbf{s} \in (\mathbb{Z}/q\mathbb{Z})^n$. The **LWE distribution** $A_{\mathbf{s},\sigma}$ is the distribution on $(\mathbb{Z}/q\mathbb{Z})^{n+1}$ defined as follows: choose uniformly at random a length n row vector $\mathbf{a}$ with entries in $\mathbb{Z}/q\mathbb{Z}$; choose an integer e randomly according to the distribution D_σ; set

$$b \equiv \mathbf{as} + e \pmod{q}$$

and consider $(\mathbf{a}, b)$.

The **learning with errors** problem (LWE) is: Given n, q, σ and an oracle which outputs values $(\mathbf{a}, b)$ drawn from the LWE distribution $A_{\mathbf{s},\sigma}$, to compute the vector **s**.

The LWE problem is well-defined in the sense that, if sufficiently many samples are drawn from the LWE distribution, one value for **s** is overwhelming more likely than the others to have been used to generate the samples.

In practice it is common to use the ring variant of the LWE problem. Let $R = \mathbb{Z}[x]/(x^N + 1)$ and $R_q = \mathbb{Z}_q[x]/(x^N + 1)$ where $N = 2^m$ is a parameter. One can interpret an element of R as a length N vector over $\mathbb{Z}$. A vector in $\mathbb{Z}^N$ with small norm is interpreted as a "small" ring element. The Ring-LWE problem is: Fix $\mathbf{s} \in R_q$, then given samples $(\mathbf{a}, \mathbf{b} = \mathbf{as} + \mathbf{e}) \in R_q^2$ where **a** is uniformly chosen in R_q and where $\mathbf{e} \in R$ is a "small" ring element (having entries chosen from some discrete Gaussian), to compute **s**. For details on Ring-LWE see Lyubashevsky, Peikert and Regev [15, 16].

It is crucial, to maintain the difficulty of the LWE problem, that the error values can occasionally be large. For example, Arora and Ge [1] and Ding [7] give approaches to attack LWE cryptosystems if the errors are known to be bounded. Hence, when one is sampling error values from the Gaussian distribution one must ensure that the tail is not truncated too severely.

3.1.1. *Regev's Cryptosystem.* The private key is a length n column vector **s** over $\mathbb{Z}/q\mathbb{Z}$, where q is prime. The public key is a pair $(\mathbf{A}, \mathbf{b})$ where **A** is a randomly chosen $m \times n$ matrix over $\mathbb{Z}_q$ with rows $\mathbf{a}_i$, and **b** is a length m column vector with entries $b_i = \mathbf{a}_i\mathbf{s} + e \pmod{q}$ such that e is sampled from the error distribution (with mean 0 and parameter σ). Lindner and Peikert [14] suggest

$$(n, m, q, \sigma) = (256, 640, 4093, 2.8).$$

This value for n is probably considered too small nowadays for most applications.

To encrypt a message $x \in \{0, 1\}$ to a user one chooses a row vector $\mathbf{u}^T \in \{0, 1\}^m$ (i.e., the entries of **u** are chosen uniformly at random in $\{0, 1\}$) and computes

$$(C_1, C_2) = (\mathbf{u}^T\mathbf{A} \pmod{q}, \mathbf{u}^T\mathbf{b} + x\lfloor q/2 \rfloor \pmod{q}).$$

Decryption is to compute $y = C_2 - C_1\mathbf{s} \pmod{q}$ and determine x by seeing if y is "closer" to 0 or $\lfloor q/2 \rfloor$. Note that decryption fails with some probability, which may be made arbitrarily small by taking q sufficiently large compared with m and σ (see Lemma 5.1 of Regev [24]). One can encrypt many bits by either repeating the above process for each bit or by using a single **A** and different values for **s** (e.g., see Lindner and Peikert [14]).

Both key generation and encryption require generating randomness. Encryption just requires generating uniform bits, which is easy using entropy external to the device or a good pseudorandom generator. However, key generation requires sampling from the Gaussian error distribution, which is not an entirely trivial computation.

There is a dual version of LWE: in this case the public key is **A** and $\mathbf{u}^T\mathbf{A} \pmod{q}$, and the ciphertext involves computing $C_1 = \mathbf{b} = \mathbf{As} + \mathbf{e} \pmod{q}$ and $C_2 = \mathbf{u}^T\mathbf{b} + x\lfloor q/2 \rfloor \pmod{q}$. The disadvantage, in practice, of this method is that the encryption algorithm now has to perform the Gaussian sampling. Hence, for constrained devices, we believe the original encryption scheme will be more practical than the dual scheme.

3.1.2. *Other encryption schemes.* Lindner and Peikert [14] and Lyubashevsky, Peikert and Regev [16] have given improved encryption schemes. We give a re-writing of the Ring-LWE variant from Section 8.2 of [16].

The public key is $(\mathbf{a}, \mathbf{b}) \in R_q^2$ (for details see [16]). To encrypt a message $\mu \in R/(p)$ (where p is a small prime) one first chooses $\mathbf{z}, \mathbf{e}', \mathbf{e}''$ from certain discrete Gaussian distributions and computes (where t and $\hat{N}$

are defined in Definition 2.17 of [16])

$$\mathbf{u} = \hat{N}(\mathbf{za} + p\mathbf{e}'), \ \ \mathbf{v} = \mathbf{zb} + p\mathbf{e}'' + t^{-1}\mu.$$

Decryption is a simple computation and rounding. The algebraic details are not important to our discussion. The important fact is that it is necessary to sample from Gaussians as part of the encryption process.

This scheme is more efficient than Regev's scheme in terms of bandwidth and number of ring operations. However, the necessity to sample from discrete Gaussians may cause other penalties to the efficiency if using constrained devices. Also note that no actual parameters are given in [16], so it is hard to assess practicality.

3.2. **Signature Schemes.** There are other applications that require sampling from the error distribution as part of the "on-line" computation. The most notable of such applications are signature schemes based on lattice problems. Since authentication is often an important security issue for constrained devices it is natural to study how practical these schemes might be on such devices. We briefly recall some of these proposals.

3.2.1. *Gentry-Peikert-Vaikuntanathan Signatures.* Gentry, Peikert and Vaikuntanathan [10] presented a signature scheme based on lattices. The scheme fits into the "hash-and-sign" paradigm. We do not give all the details, but the basic scheme is as follows. The public key consists of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ defining a rank m lattice $\Lambda^{\perp}(\mathbf{A}) = \{\mathbf{e} \in \mathbb{Z}^m : \mathbf{Ae} \equiv 0 \pmod{q}\}$. The private key consists of a "short" full rank set of vectors in this lattice that allows one to invert the function $f(\mathbf{e}) = \mathbf{Ae} \pmod{q}$. The signature on message m is a "short" vector $\mathbf{s} \in \mathbb{Z}^m$ such that $\mathbf{As} \equiv H(m) \pmod{q}$, where H is a cryptographic hash function (random oracle) mapping to $\mathbb{Z}_q^n$.

The signing algorithm of [10] is as follows. The algorithm first computes an arbitrary preimage $\mathbf{t} \in \mathbb{Z}^m$ such that $\mathbf{At} \equiv H(m) \pmod{q}$ then finds a lattice vector $\mathbf{v} \in \Lambda^{\perp}(\mathbf{A})$ "close" to $\mathbf{t}$, so that the signature to be returned is $\mathbf{t} - \mathbf{v}$. The instance of the closest vector problem is solved using the "nice" private lattice basis and a variant of the Babai nearest plane algorithm. But instead of rounding a Gram-Schmidt coefficient to the nearest integer, so as to choose the nearest plane at each stage in the algorithm, the rounding is performed using a discrete Gaussian distribution. This rounding is performed by the SampleD function of Section 4.2 of [10]. This function involves sampling m integers from the distribution $D_{\mathbb{Z},\sigma,c}$ for *varying* values of σ and c (the parameter σ depends on the norm of the i-th Gram-Schmidt vector while the "center" c depends on the current Gram-Schmidt coefficient that is being rounded).

The approach in [10] for actually sampling these integers is to use rejection sampling: One samples $x \in \mathbb{Z}$ uniformly from a large interval $[-\sigma \log(n), \sigma \log(n)]$ (where n is a security parameter) of integers and then accepts with probability proportional to $\rho_{\sigma,c}(x)$ (i.e., the probability that this integer x would be output). This method is inefficient in several ways, as we now discuss.

One particular inefficiency is that one needs to sample from the interval many times before acceptance. We now estimate the expected number of trials for each value. Suppose we sample from the uniform distribution on $[-20\sigma, 20\sigma]$ and wish to generate a discrete Gaussian given by $\rho_{\sigma,c}(x)$. We take $c = 0$ for simplicity. Using notation as in Section 2.1, we have $g(x) = 1/(1 + 40\sigma)$ and $f(x) = e^{-x^2/(2\sigma^2)}/(\sqrt{2\pi}\sigma)$ giving $f(0) = 1/(\sqrt{2\pi}\sigma)$ and so $M_R = (1 + 40\sigma)/(\sqrt{2\pi}\sigma) \approx 40/\sqrt{2\pi} \approx 16$. Hence, around 16 uniformly chosen integers must be generated and around 16 computations of the function $\rho_{\sigma,c}(x)$ are needed for each sample.

Another issue with rejection sampling is that one needs random bits as input for two different purposes. Recall that the algorithm first generates an integer a uniformly at random from $[-\sigma \log(n), \sigma \log(n)]$, then accepts it with probability p. The rejection is done by sampling from a two-element set with probabilities p and $1-p$ and making a decision based on the element that is generated. One can use the Knuth-Yao algorithm (see Section 5) for this latter task.[1] Finally, since σ and c vary, we need to compute the probabilities $\rho_{\sigma,c}(x)$ for various x, which requires high precision floating-point arithmetic.

Putting all the above together, the number of uniform input bits used for sampling from the discrete Gaussian $D_{\sigma,c}$ using rejection sampling will be around $16(\log_2(40\sigma) + 2) \approx 117 + 16\log_2(\sigma)$ rather than the value $\log_2(2\pi e\sigma^2)/2 \approx 2 + \log_2(\sigma)$ of the entropy. For these reasons, and others, we believe the signature scheme using this approach is completely impractical on constrained devices (or perhaps almost any device).

3.2.2. *The improvements by Peikert.* Peikert [22] has given an alternative method to address the closest vector problem. He proposes a variant of the Babai rounding algorithm, again choosing the integers from a discrete Gaussian distribution. One crucial difference is that, in Peikert's method, the parameter s is fixed and only the center changes.

Section 4.1 of [22] discusses the basic operation of sampling from the distribution $D_{\mathbb{Z},\sigma,v}$ (where, without loss of generality $0 \le v < 1$). Peikert uses the inversion method. More precisely, one chooses a multiplier $M = \omega(\sqrt{\log(n)})$ such that one can restrict to sampling integers z in the range $v - \sigma M \le z \le v + \sigma M$ (typical values for M are between 20 and 40). One then computes a table of cumulative probabilities $p_z = \Pr(D_{\sigma,v} \le z)$. To sample from the distribution one then generates a uniformly chosen real number a in the interval $[0, 1]$ (by generating its binary expansion as a sequence of uniformly chosen bits) and returns the integer z such that $p_{z-1} < a \le p_z$.

One serious issue with this approach is the size of the table and the cost of computing it. The standard goal for good quality theoretical results is to be able to sample from a distribution whose statistical difference with the desired distribution is around 2^{-90} to 2^{-100}. We take some parameters from Figure 2 of Micciancio and Peikert [19] (the parameters for GPV-signatures would be at least as bad as these, so our calculation will be a lower bound on the actual size of the table). They suggest $n = 13812$ and "$s = 418$" (which corresponds to $\sigma \approx 167$). So it is required to compute a large table containing approximations (up to accuracy ϵ) to the probabilities for each integer in the range $[-M\sigma, M\sigma]$. By Lemmas 1 and 2 we could take $20 \le M \le 40$). and ensure that $2nM\sigma\epsilon < 2^{-100}$ which means taking $\epsilon \approx 2^{-126}$ (since $26 < \log_2(2nM\sigma) < 28$). In other words, we need a table with $5,000 \le 2 \cdot M \cdot \sigma \le 12,000$ entries, each with around 126 binary digits of fixed-point accuracy. Such a table requires around 100 kilobytes. If the entries were all centered on 0 then one could exploit symmetry and use with a table of $342,972$ entries and using 42 kilobytes. These values are potentially feasible for some devices, but not ideal.

3.2.3. *Micciancio-Peikert signatures.* Micciancio and Peikert [19] give a variant of GPV signatures with numerous practical advantages. It is beyond the scope of our paper to recall the details of their scheme, but we discuss some crucial elements.

The main part of the signing algorithm is the $\mathsf{SampleD}$ procedure given as Algorithm 3 of [19]. This algorithm can be broken down into an "offline" phase and an "online" phase. The difference is that the offline phase does not depend on the message, and so this computation can be performed in advance during idle time of the device, or a certain number of such values may be precomputed and stored on the device for one-time use. Amazingly, the online phase is permitted to be deterministic, and if there is sufficient storage on the device then any random sampling steps may be replaced by table look-ups. Hence, from the point of view of requiring random input bits to the algorithm, the online phase of the Micciancio-Peikert scheme is perfect.

We now give further details of the online phase as explained in Section 4.1 of [19]. The basic task is: Given $\mathbf{g} \in \mathbb{Z}_q^k$ and $u \in \mathbb{Z}_q$ to return a vector $\mathbf{x} \in \mathbb{Z}^k$ such that $\mathbf{x} \cdot \mathbf{g} \equiv u \pmod q$ and such that $\mathbf{x}$ is sampled

[1] As will be discussed later, even if $\Pr(a)$ is very small then this only requires around two random input bits on average. But we do need to *calculate* the binary tree for the Knuth-Yao algorithm for each case, which seems inconvenient.

from a discrete Gaussian distribution on $\mathbb{Z}^k$ with small parameter $2r \approx 9$. One approach is to store a table of q values containing one such $\mathbf{x} \in \mathbb{Z}_q^k$ for each value of u. However, the storage requirements for the table for the online phase are as follows: we need q vectors in $\mathbb{Z}_q$ of length k; taking the values from Figure 2 of [19] gives $q = 2^{24}$ and $k = 24$, giving a total of $qk \log q \approx 2^{33}$ bits or 1.1 gigabytes, which is not practical.[2] A second approach is to use the fact that we have a nice basis and can solve CVP. This approach requires sampling from either $2\mathbb{Z}$ or $2\mathbb{Z}+1$ with discrete Gaussian distribution and small parameter $s \approx 9$. This can be performed in practice by precomputing many such values and using each of them once only. Section 4.1 of [19] also suggests a hybrid algorithm.

The offline phase requires sampling perturbations from a spherical Gaussian distribution on a certain lattice. Section 5.4 of [19] suggests this can be done by sampling a continuous non-spherical Gaussian and then taking randomised-rounding using samples from $D_{\mathbb{Z},r}$ with small parameter $r \approx 4.5$. These operations still require high-precision floating-point arithmetic or large precomputed tables.

Overall, it seems that the signature scheme of [19] is not ideally-suited for constrained devices. But one could consider a version of it for devices with relatively low storage as follows. The samples needed for the offline phase could be precomputed and stored on the device (limiting the number of signatures produced by the device during its lifetime). The online phase could be performed, less quickly than the approach in [19], using an algorithm to efficiently sample from $2\mathbb{Z}$ and $2\mathbb{Z}+1$ with small parameter, such as the ones we will discuss later in our paper (this requires precomputed tables of modest size). Hence, it seems that our methods may be useful for the future development of the Micciancio-Peikert scheme.

3.3. **Lyubashevsky Signatures.** Lyubashevsky and his co-authors have developed secure lattice-based digital signature schemes based on the Fiat-Shamir paradigm. In other words, the signature behaves like a proof of knowledge of the private key. A crucial technique is the use of rejection sampling to ensure that the distribution of signatures is independent of the private key. The distributions used are sometimes Gaussians and sometimes uniform distributions. We briefly recall some of their schemes now.

The signature scheme of Lyubashevsky [17] does not seem to require sampling from discrete normal distributions. Instead, the signing algorithm requires sampling coefficients of a ring element from a uniform distribution. This scheme is therefore practical to implement on a constrained device. However, the signatures are very large (at least 49,000 bits), which is not good for constrained devices.

Lyubashevsky [18] gives signature scheme for which the signatures are around 20,500 bits. The scheme in [18] requires sampling from a discrete Gaussian distribution on $\mathbb{Z}^m$ with parameter σ being very large (between $3\cdot 10^4$ and $1.7\cdot 10^5$). The corresponding values for m are between 1024 and 8786; for the remainder we consider $(m, \sigma) = (1024, 1.7 \cdot 10^5)$. The distribution is centered on the origin in $\mathbb{Z}^m$, so there are no issues about changing the center. Precisely, one is required to sample a length m vector $\mathbf{y} \in \mathbb{Z}^m$ according to $D_{\mathbb{Z}^m,\sigma}$, and this can be done by taking m independent samples from D_σ for the m entries in the vector. Lyubashevsky suggests to use rejection sampling to generate $\mathbf{y}$. Rejection sampling is also used by the signing algorithm to ensure that the signature is independent of the secret. But at least the value σ is fixed for all samples and so one can easily make use of precomputed tables.

Suppose we use Peikert's inversion method [22] in this application (with $m = 1024$ and $\sigma = 1.7 \cdot 10^5$). Again applying Lemma 1 we need 124 bits of precision (and hence storage) for each probability. We therefore need a table with $M \cdot \sigma \geq 3.4 \cdot 10^6$ entries (for $20 \leq M \leq 40$). This table can be precomputed and stored on the device, removing any need for floating-point arithmetic to be implemented, which is a good feature. The problem is that the table needs at least 50 megabytes of storage, which is not available on a constrained device.

[2]Since the parameter is 9 and so $\sigma = 3.6$ we will actually only generate integers in the range $|x_i| < 12\sigma \approx 43$. Hence only 7 bits are needed to represent each entry of the vector and the storage can be reduced to about 0.3Gb.

Güneysu, Lyubashevsky and Pöppelmann [11] gave a signature scheme that only requires uniform distributions, and for which signatures are around 9,000 bits. The scheme is easily implemented on a constrained device, however its security is based on non-standard assumptions and a full proof of security is not given in [11]. Following their work, Bai and Galbraith [2] have given a provably secure scheme that can be implemented using uniform distributions and has signatures of size between 9,000 and 12,000 bits.

Ducas, Durmus, Lepoint and Lyubashevsky [9] have given a new scheme with several further tricks to reduce the signature size. Their scheme uses discrete Gaussian distributions (indeed, bimodal distributions) and the signing algorithm requires sampling from a discrete Gaussian with parameter $\sigma = 107$. Their paper includes some very interesting new techniques to sample from discrete Gaussians (their approach does not require a very large amount of precomputation, nor floating-point arithmetic). The security is based on a non-standard computational assumption related to NTRU, and the scheme has signatures of around 5,000 bits.

To conclude, the schemes of [9, 11, 2] seem to be quite feasible for a constrained device, whereas the scheme in [18] seems to be infeasible to implement using the Knuth-Yao method. Further, the hash-and-sign signature schemes seem to be impractical on constrained devices, however there is some hope to make the Micciancio-Peikert scheme practical using methods like those discussed in our paper.

4. Computing Probabilities

We now back-up a little and ask precisely what resources are required to compute these probabilities and distributions. In particular, how is the function $\exp(x)$ computed to high precision anyway?

4.1. Computing the Exponential Function.

We now discuss the standard algorithms for computing $\exp(x)$ where x is a real number (we are mainly interested in the case when $x < 0$).

Recall that the binary floating-point representation of a real number x is as a sign bit s, a "significand" or "mantissa" m and a signed exponent e such that

$$x \approx (-1)^s 2^e (1.m)$$

i.e., the bits m are the binary expansion after the decimal point where the leading bit is always 1 (since otherwise one can set $e = e + 1$). An alternative is for m to be interpreted as an integer and the real number is $(-1)^s 2^e (1 + m)$. The usual settings for 32-bit floating-point numbers are for the significand to require 23 bits (representing a 24-bit integer due to the leading 1) and an 8 bit signed exponent. Double precision (i.e., 64-bit) floating-point representation has a 53-bit significand and a 10-bit signed exponent.

The values 2^n, where $n \in \mathbb{Z}$, $-127 \leq n \leq 128$ therefore have an exact floating-point representation with $e = n$ and $m = 0$. We now explain the standard method to compute an approximation to 2^y for $y \in \mathbb{R}$: One sets $e = \lfloor y \rceil$ and $z = e - y$ so that $|z| \leq 1/2$ and computes an approximation to the binary expansion of 2^z. This can be done by using precomputed tables with linear interpolation (for a survey of such methods see [4, 6]).

The trick to computing floating-point approximations to $\exp(x)$ for $x \in \mathbb{R}$ is to note that $\exp(x) = 2^{x/\ln(2)}$, where $\ln(2)$ is the log to base e of 2. Hence, to compute $\exp(x)$ one first computes $y = x/\ln(2) \in \mathbb{R}$ and then computes a floating-point approximation to 2^y as above.

There are two variants of the above method, both of which involve translating the computation 2^y for $|y| \leq 1/2$ or $0 < y < 1$ back to $\exp(x)$ as $2^y = \exp(y \ln(2))$. The first is to use the power series expansion $\exp(x) = 1 + x + x^2/2! + x^3/3! + \cdots$ by summing an appropriate number of terms (see [21]). This avoids storing large tables, but adds to the running time.

A second variant for computing $\exp(x)$ for small $x \geq 0$, due to Specker [26], is to use the fact that $\exp(x) = \prod_{i=0}^{R} b_i$ if and only if $x = \sum_{i=0}^{R} \ln(b_i)$. Specker suggests to choose each b_i to be either 1 or

$1 + 2^{-i}$, so that $\ln(b_i)$ is either 0 or approximately 2^{-i}. It is relatively easy to decide which value of b_i to choose at each step by looking at the binary expansion of $x - \sum_{j=0}^{i-1} \ln(b_j)$. One could precompute the values of $\ln(b_j)$, or compute them "on the fly" using power series or tables. This approach gives a tradeoff between running time and storage requirements. A related method, using the arithmetic-geometric mean, is presented in Section 5.5.3 of Muller [20].

To summarise, there are several methods in the literature to compute high-precision floating-point approximations to $\exp(x)$. All use either large precomputed tables or else a large number of floating-point operations (e.g., to sum a power series). None of these methods are particularly suited to constrained devices.

4.2. **Computing the Discrete Gaussian Distribution.** We are mainly interested in the distribution $D_{\sigma,c}$ on $\mathbb{Z}$ defined in Section 2. Recall that $\sigma, c \in \mathbb{R}$, $\rho_{\sigma,c} = \exp(-(x-c)^2/(2\sigma^2))$,

$$S_{\sigma,c} = \rho_{\sigma,c}(\mathbb{Z}) = \sum_{x=-\infty}^{\infty} \rho_{\sigma,c}(x)$$

and $D_{\sigma,c}$ is defined to be the distribution on $\mathbb{Z}$ such that the probability of $x \in \mathbb{Z}$ is $\rho_{\sigma,c}(x)/S_{\sigma,c}$. To compute this probability function it is necessary to compute $S_{\sigma,c}$. When $c = 0$ we expect $S_\sigma \approx \sqrt{2\pi}\sigma$. Indeed, for even relatively moderate values for σ (e.g., $\sigma > 5$) and then the formula $\sqrt{2\pi}\sigma$ is actually accurate enough for our purposes.

When $c \neq 0$ then one cannot use a simple formula, however when σ is sufficiently large then the approximation $\sqrt{2\pi}\sigma$ remains good enough. If σ is small then to sample from $D_{\sigma,c}$ one needs to compute a close approximation to the value $S_{\sigma,c}$, for example as

$$\sum_{x=-12\sigma}^{12\sigma} \rho_{\sigma,c}(x).$$

5. THE KNUTH-YAO ALGORITHM

Knuth and Yao [13] developed an algorithm to sample from non-uniform distributions using as few uniform input bits as possible. In other words, the aim of the Knuth-Yao algorithm is to sample using a number of input bits that is as close as possible to the minimum value given by the entropy of the distribution. A good reference is Chapter 15 of Devroye [5].

The method is quite simple. Suppose we want to sample from the set $\{1, \ldots, m\}$ such that an integer a is sampled with probability p_a. Write each p_a in binary. Now write down a binary tree (when implementing the algorithm one uses a table) called a discrete distribution generating (DDG) tree. At each level we will choose some vertices to be internal (labelled as "I"), and some to be leaves. The number of leaves at level i is the number of values for a such that the i-th decimal digit of p_a is a one. Each leaf is labelled with a different value for a. To sample from the distribution one walks down the tree from the root using one uniform bit at each step to decide which of the two children to move to. When one hits a leaf one outputs the integer label for that leaf.

For example, let $p_1 = 0.10010$, $p_2 = 0.00011$, $p_3 = 0.01011$ be the probabilities of choosing elements from the set $\{1, 2, 3\}$. The tree has one leaf at level 1 (labelled 1); one leaf at level 2 (labelled 3); no leaves at level 3; three leaves at level 4 (labelled 1, 2 and 3); and finally two leaves at level 5 (labelled 2 and 3). The tree is drawn in Figure 2.

In general, the binary tree will be infinite. It is easy to see that the Knuth-Yao algorithm samples from the correct distribution, even if the probabilities have infinite binary expansions (though we will only use it with finite expansions).

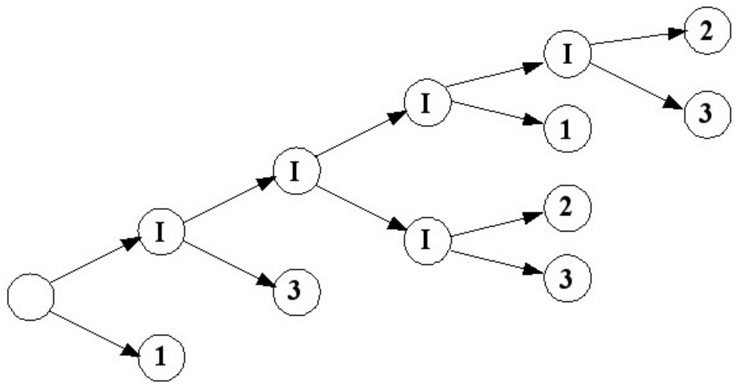

FIGURE 2. Knuth-Yao DDG tree.

Level	1	2	3	4	5
Nodes	I	I	I	I	2
	1	3	I	1	3
				2	
				3	

FIGURE 3. Tabular representation of Knuth-Yao DDG tree.

It can also be shown that the expected number of uniformly generated input bits used in the algorithm is at most two more than the entropy of the distribution (see Theorem 2.1 and the Corollary to Theorem 2.2 of Knuth and Yao [13] or Theorems 15.2.1 and 15.3.1 of Devroye [5]). An easy exercise is to consider sampling from $\{0, 1\}$ where the probability of 0 is p and the probability of 1 is $1 - p$. If $p = 1/2$ then the expected number of uniform bits used by the algorithm is 1, if $p = 1/4$ then the expected number of bits used is 1.5. More generally, the entropy is between 0 and 1 and the expected number of trials is less than 3, and is close to 2 when p is very small.

Suppose that the probabilities all have at most k bits in their binary expansion, so that the tree has depth k. The number of leaves of the tree is equal to the number of ones in all m binary expansions, so the tree has at most mk leaves. A tree with mk leaves has $mk - 1$ internal vertices, so the tree requires $O(mk)$ space to represent.

It is not necessary to represent the algorithm using a tree. Instead, one can store the information as a table as in Figure 3. The table itself is more-or-less just a re-writing of the binary expansions of the probabilities, though note that the i-th column may contain as many as 2^i entries (as long as $2^i < mk$, since that is a bound on the total number of leaves and internal vertices). It is not necessary to store the entire table to perform the algorithm. Instead, one just needs to store the probabilities and a single column of the table; from this information it is simple to construct the next column of the table (more details of this process are given in [25]). The Knuth-Yao algorithm can be organised so that the random walk down the tree only makes use of this "local" information.

5.1. **The Knuth-Yao method for sampling discrete Gaussians.** We consider using the Knuth-Yao method for sampling discrete Gaussians. We now suppose we are sampling from D_σ. One only needs to sample from $\mathbb{Z}_{\geq 0}$ and use a uniform bit to choose the sign (in this setting one has to set the probability of zero to be half its correct value since $-0 = +0$ and so 0 is sampled twice).

When using the Knuth-Yao algorithm we need to store the binary expansions of the probabilities. Hence, one might think that the algorithm requires more-or-less the same storage as Peikert's method (which stores a table of cumulative probabilities). However, the Knuth-Yao table has many leading zeroes and there is no reason to store them directly. So one can expect to reduce the storage for the table.

For example, suppose $\sigma = 10$ and that we need the probabilities $\rho_\sigma(x)$ for $x \in [0, 200]$. We list in the below table the values $p_\sigma(i\sigma)$ for $i = 0, 1, 2, 3, 4, 5, 6, 7$ with 20 bits of precision. One might assume that the table requires $201 \cdot 20 = 4020$ bits of storage, however it is clear that the first 4 bits are always zero and so do not need to be stored. Furthermore, once past two standard deviations the first 7 bits are always zero, and past three standard deviations the first 11 bits are all zero, and so on. Just storing the data in individual tables in "blocks" of 10, each with a different precision, leads to storage of 5 small integers (the number of leading zero bits) together with $11 \cdot 16 + 10 \cdot 15 + 10 \cdot 13 + 10 \cdot 9 + 20 \cdot 4 = 626$ bits. This idea has subsequently been developed in [25].

x	$p_\sigma(x)$	Binary expansion of $p_\sigma(x)$
0	0.01994711	0.00000101000110110100
1	0.03969525	0.00001010001010010111
10	0.02419707	0.00000110001100011100
20	0.00539909	0.00000001011000011101
30	0.00044318	0.00000000000111010001
40	0.00001338	0.00000000000000001110
50	0.00000015	0.00000000000000000000

The above remarks indicate how the table of probabilities can be compressed by omitting some leading zeroes. We now consider the Knuth-Yao algorithm itself. One approach is to take a large set $[-M\sigma, M\sigma]$ for $20 \leq M \leq 40$ and apply the Knuth-Yao method for all the probabilities together. If one stores the current column of the Knuth-Yao table then the number of entries to be stored may be very large. Algorithm 1 of [25] gives an approach that does not require storing any columns of the Knuth-Yao table. An alternative, that may have other advantages, is to work with blocks.

5.2. **Knuth-Yao in blocks.** The idea is to partition $[0, M\sigma]$ (e.g., $20 \leq M \leq 40$) as a union of pair-wise disjoint sets $A_1, \dots, A_t$ (being the "blocks") such that the probability $\rho_\sigma(A_i)$ is roughly the same for all $1 \leq i \leq t$. One stores high-precision approximations p_{A_i} to $\Pr(A_i)$ and performs the Knuth-Yao algorithm in two stages. The first stage is to determine in which set A_i the value will be chosen. For the second stage one stores approximations to the conditional probabilities $\Pr(a|A_i)$ and performs the Knuth-Yao algorithm to decide the value a to be sampled. One could employ more than two iterations of the idea.

For example, suppose $\sigma = 50$ and we are sampling from $[0, 40\sigma]$. Taking $t = 10$ we find the following intervals all having similar weight in the distribution:

$$A_1 = [0, 12],\ A_2 = [13, 25],\ A_3 = [26, 39],\ A_4 = [40, 54],\ A_5 = [55, 70],\ A_6 = [71, 88],$$
$$A_7 = [89, 109],\ A_8 = [110, 136],\ A_9 = [137, 180],\ A_{10} = [181, 2000].$$

For example, we have $\Pr(A_1) \approx 0.102647$ (this is normalised to sampling from $[0, 20\sigma]$) and $\Pr(A_2) \approx 0.100995$ and $\Pr(A_{10}) \approx 0.070511$.

Note that the statistical distance of the sampled distribution from the desired distribution is at least as big as

$$\frac{1}{2} \sum_{i=1}^{t} |p_{A_i} - \Pr(A_i)|$$

k	3	4	5	6	7	8	9	10
$-\lfloor \log_2(p_k) \rceil$	24	29	36	44	53	64	76	90

FIGURE 4. Number of leading zeroes in binary expansion of tail probabilities.

and so it is necessary that the values p_{A_i} be computed to very high precision. However, one does not need such a high precision for the terms $\Pr(a|A_i)$, since $\Pr(a) = \Pr(a|A_i)\Pr(A_i)$. This is essentially another way of viewing the "omission" of leading zero bits in the probability.

Since we are no longer performing the exact Knuth-Yao algorithm it is necessary to consider whether the number of input bits used in the sampling is increased significantly. One can verify that this is not a serious issue. To support this we performed some small experiments. For example, taking $\sigma = 2.8$, the entropy of the distribution is around 2.45 and expected number of bits to sample using the Knuth-Yao algorithm is computed to be approximately 4.44, which agrees with the theoretical upper bound of $2.45 + 2 = 4.45$. Now consider the variant of the Knuth-Yao algorithm where one first chooses a sign bit and then samples from $[0, M\sigma]$ rather than $[-M\sigma, M\sigma]$ (we only go as far as $M = 12$ since we are not working with extremely high precision). One might expect the number of bits needed to be increased by one, but the reduced sample space compensates for this. The entropy of the "half" distribution is approximately 1.85, and adding the sign bits gives a minimum of 2.85 bits for sampling using this method. We calculated the expected number of bits used by the Knuth-Yao algorithm to be approximately 4.63. So the additional penalty for halving the size of the table is only about 0.2 bits on average.

Similarly, using the block method instead of the full table of probabilities only adds a small fractional amount to the expected number of bits required. For the case $\sigma = 2.8$ one might first partition the range $[0, 33]$ into the four intervals $\{0\}, \{1\}, \{2, 3\}$ and $[4, 33]$. The respective probabilities are roughly 0.142, 0.267, 0.381 and 0.209. If either of the last two sets is chosen then the Knuth-Yao algorithm is repeated with an appropriate table to choose the actual integer sampled. We computed that the expected number of bits used by this variant of the algorithm is approximately 5.23, which is about 0.6 bits more than the previous method – not a large price to pay.

5.3. **Constructing the tail on the fly.** One might not want to precompute and store all the probabilities in the final block, which corresponds to the long tail. So we now sketch an approach to compute this as required.

Our observation is that, since the tail probabilities are all extremely small, they do not need to be computed to such high floating-point precision. Hence, it may be possible to compute approximations to the probabilities of the tail values on-the-fly on a constrained device using relatively standard floating-point precision.

Figure 4 gives the number of leading zero bits in the probabilities $p_k = \exp(-(k\sigma)^2/(2\sigma^2))/S_\sigma$ for $\sigma = 10^5$ and various values for k. One sees that to achieve 120 significant bits of fixed-point precision (i.e., $\epsilon = 1/2^{120}$) it is only necessary to store $\exp(-x^2/(2\sigma^2))/S_\sigma$ to 30 bits of precision when $x \geq 10\sigma$. Hence, it is sufficient to use standard double-precision (64-bit) floating-point arithmetic to compute the tail probabilities for $x > 8.5\sigma$.

Hence, it is of interest to consider methods for generating a list of approximations to the probabilities $\exp(-x^2/(2\sigma^2))/S_\sigma$ when x is large. As already mentioned in Section 4.1, it is not necessarily trivial to compute the $\exp(x)$ function using numerical methods. The trick is to exploit the fact that we want to compute an entire list of values. Of course, the computation cost is linear in the length of the list, which may be very significant when σ is large. Hence, it is likely that a partial list is computed first and the extreme values only computed if absolutely necessary.

We now explain how to use a recurrence to compute a list of values for $\exp(-x^2/2\sigma^2)$, hence avoiding the methods in Section 4.1. Let $A(x) = \exp(-x^2)$ and $B(x) = \exp(-x)$. Then

$$A(x+1) = cA(x)B(x)^2 , \qquad B(x+1) = cB(x)$$

where $c = \exp(-1)$. If the initial values $A(x_0), B(x_0)$ and c are precomputed and stored on the device then a list of values $A(x_0 + i), B(x_0 + i)$ can therefore be computed relatively easily using floating-point arithmetic to the required precision.

6. Using the Central Limit Theorem

We have explored some methods that may be appropriate when σ is rather small, but they seem to require prohibitive storage when σ is large. The aim of this section is to sketch an alternative approach for the case when σ is large. Unfortunately this does not seem to lead to a practical solution.

Recall the central limit theorem: If $X_1, \dots$ are random variables on $[-M, M]$ (e.g., uniformly distributed) with mean 0 and variance σ^2 then

$$S_k = \tfrac{1}{k} \sum_{i=1}^{k} X_i$$

is a random variable on $\mathbb{R}$ with mean 0 and variance $k\sigma^2$. To sample from a discrete Gaussians one could generate a large number k of integers $x_1, \dots, x_k$ uniformly distributed in a range $[-M, M] \cap \mathbb{Z}$ and then return their sum $\sum_{i=1}^{k} x_i$. It follows from the central limit theorem that, for sufficiently large k, the sum is "close" to a discrete normally distributed integer random variable with variance $kM^2/3$. Hence, one chooses k so that the variance is the value desired for the application. The hope is that this can lead to an algorithm for sampling from discrete Gaussians that does not require floating-point arithmetic or large tables.

One simple case is to sample the integers x_i from $\{-1, 1\}$, but this leads to the unfortunate problem that the sampled value has the same parity as k. Other natural choices are $\{-1, 0, 1\}$ or $\{-2, -1, 1, 2\}$.

The main problem with this approach is to get a sufficiently small statistical difference with the true distribution. For example, taking $\sigma = 20$ and $k = 160$ sampling uniformly from $\{-2, -1, 1, 2\}$ gives statistical difference approximately $0.000612 \approx 1/2^{10.7}$. Of course, one can scale the problem: In other words, sample as above for a large multiple Mk of the expected number of samples and then scale the results by grouping them in chunks (essentially, dividing by M and rounding the distribution on $\frac{1}{M}\mathbb{Z}$ to $\mathbb{Z}$). However, the rate of convergence in the central limit theorem is slow and so this approach does not seem to be practical to generate distributions within a statistical difference of 2^{-100} with a true discrete Gaussian.

Another drawback is that the method still needs a large number of uniform bits as input: typically one needs to take $k \approx \sigma$ samples, so the number of bits used is proportional to σ, rather than proportional to $\log(\sigma)$ as desired from the entropy calculations in Section 2.

7. Recent Literature

Subsequent to the submission of our paper, several authors have worked on this problem:

(1) Buchmann, Cabarcas, Göpfert, Hülsing and Weiden [3] give a different approach to sampling, based on the ziggurat method, that combines precomputed tables and rejection sampling. However, the rejection sampling can be made very efficient. Large precomputed tables are still required.

(2) Sinha Roy, Vercauteren and Verbauwhede [25] build on our work, giving some improvements to the storage and a low-level implementation of the method.

(3) Ducas and Nguyen [8] discuss sampling from discrete Gaussians on lattices using m-bit floating-point precision. Their approach uses rejection sampling to sample from a discrete Gaussian on $\mathbb{Z}$ and assumes a floating-point implementation of the $\exp(x)$ function.

(4) Ducas, Durmus, Lepoint and Lyubashevsky [9] give a different way to efficiently compute probabilities without large precomputation. Section 6.2 of [9] shows how to do rejection sampling by constructing the exact probabilities on the fly without large precomputation. Section 6.3 of [9] gives a method to easily sample from an exponential distribution that is already close to a discrete Gaussian, and then uses rejection sampling so that the ouput is close to the desired discrete Gaussian.
(5) Karney [12] samples from the exponential distribution using the von Neumann method and then uses rejection sampling to correct to a Gaussian distribution. Hence, the method is very similar to that in [9]. The paper is written for continuous Gaussians but it seems it can also be used for discrete Gaussians. The method can be implemented without needing floating-point arithmetic.

ACKNOWLEDGEMENTS

We thank Mark Holmes, Charles Karney, Vadim Lyubashevsky and Frederik Vercauteren for comments and corrections.

REFERENCES

1. S. Arora and R. Ge, New Algorithms for Learning in Presence of Errors, in L. Aceto, M. Henzinger and J. Sgall (eds.), ICALP 2011, Springer LNCS 6755 (2011) 403–415.
2. S. Bai and S. D. Galbraith, An Improved Compression Technique for Signatures Based on Learning with Errors, in J. Benaloh (ed.), CT-RSA 2014, Springer LNCS 8366 (2014) 28–47.
3. 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, to appear in proceedings of SAC 2013.
4. J. Detrey and F. de Dinechin, Table-based polynomials for fast hardware function evaluation, in Application-specific Systems, Architectures and Processors (ASAP 2005), IEEE (2005) 328–333.
5. L. Devroye, Non-Uniform Random Variate Generation, Springer-Verlag, New York (1986)
Available from: `http://www.nrbook.com/devroye/`
6. F. de Dinechin and A. Tisserand, Multipartite table methods, IEEE Transactions on Computers, Vol. 54, No. 3 (2005) 319–330.
7. J. Ding, Solving LWE problem with bounded errors in polynomial time, eprint 2010/558, 2010.
8. L. Ducas and P. Q. Nguyen, Faster Gaussian Lattice Sampling Using Lazy Floating-Point Arithmetic, in X. Wang and K. Sako (eds.), ASIACRYPT 2012, Springer LNCS 7658 (2012) 415–432.
9. L. Ducas, A. Durmus, T. Lepoint and V. Lyubashevsky, Lattice Signatures and Bimodal Gaussians, in R. Canetti and J. A. Garay (eds.), CRYPTO 2013, Springer LNCS 8042 (2013) 40–56.
10. C. Gentry, C. Peikert and V. Vaikuntanathan, Trapdoors for Hard Lattices and New Cryptographic Constructions, in C. Dwork (ed), STOC 2008, ACM (2008) 197–206.
11. T. Güneysu, V. Lyubashevsky and T. Pöppelmann, Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems, in E. Prouff and P. Schaumont (eds.), CHES 2012, Springer LNCS 7428 (2012) 530–547.
12. C. F. F. Karney, Sampling exactly from the normal distribution, arXiv:1303.6257 (2013).
13. D. E. Knuth and A. C. Yao, The complexity of non uniform random number generation, in J. F. Traub (ed.), Algorithms and Complexity, Academic Press, New York (1976) 357–428.
14. R. Lindner and C. Peikert, Better key sizes (and attacks) for LWE-based encryption, in A. Kiayias (ed.), CT-RSA 2011, Springer LNCS 6558 (2011) 319–339.
15. V. Lyubashevsky, C. Peikert and O. Regev, On Ideal Lattices and Learning with Errors over Rings, in H. Gilbert (ed.), EUROCRYPT 2010, Springer LNCS 6110 (2010) 1–23.
16. V. Lyubashevsky, C. Peikert and O. Regev, A Toolkit for Ring-LWE Cryptography, in T. Johansson and P. Q. Nguyen (eds.), EUROCRYPT 2013, Springer LNCS 7881 (2013) 35–54.
17. V. Lyubashevsky, Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures, in M. Matsui (ed), ASIACRYPT 2009, Springer LNCS 5912 (2009) 598–616.
18. V. Lyubashevsky, Lattice Signatures without Trapdoors, in D. Pointcheval and T. Johansson (eds.), EUROCRYPT 2012, Springer LNCS 7237 (2012) 738–755.
19. D. Micciancio and C. Peikert, Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller, in D. Pointcheval and T. Johansson (eds.), EUROCRYPT 2012, Springer LNCS 7237 (2012) 700–718.
20. J.-M. Muller, Elementary Functions, Algorithms and Implementation (2nd ed.), Birkhauser, Boston, 2005.

21. F. W. J. Olver, D. W. Lozier, R. F. Boisvert and C. W. Clark, NIST Handbook of Mathematical Functions, Cambridge University Press, 2010.
22. C. Peikert, An Efficient and Parallel Gaussian Sampler for Lattices, in T. Rabin (ed.), CRYPTO 2010, Springer LNCS 6223 (2010) 80–97.
23. O. Regev, On lattices, learning with errors, random linear codes, and cryptography, STOC 2005, ACM (2005) 84–93.
24. O. Regev, On lattices, learning with errors, random linear codes, and cryptography, *Journal of the ACM*, 56(6), article 34, 2009.
25. S. Sinha Roy, F. Vercauteren and I. Verbauwhede, High precision discrete gaussian sampling on FPGAs, to appear in proceedings of SAC 2013.
26. W. H. Specker, A Class of Algorithms for $\ln x$, $\exp x$, $\sin x$, $\cos x$, $\tan^{-1} x$, and $\cot^{-1} x$, IEEE Transactions on Electronic Computers, Vol. EC-14 , Issue 1 (1965) 85–86.

E-mail address: `nagarjuncd@gmail.com`

Indian Institute of Technology, Guwahati, India.
E-mail address: `S.Galbraith@math.auckland.ac.nz`

Mathematics Department, University of Auckland, New Zealand.