Faster Gaussian Lattice Sampling Using Lazy Floating-Point Arithmetic

Léo Ducas, Phong Q. Nguyen

Faster Gaussian Lattice Sampling using Lazy Floating-Point Arithmetic

Léo Ducas and Phong Q. Nguyen

[1] ENS, Dept. Informatique, 45 rue d'Ulm, 75005 Paris, France.
http://www.di.ens.fr/~ducas/
[2] INRIA, France and Tsinghua University, Institute for Advanced Study, China.
http://www.di.ens.fr/~pnguyen/

Abstract. Many lattice cryptographic primitives require an efficient algorithm to sample lattice points according to some Gaussian distribution. All algorithms known for this task require long-integer arithmetic at some point, which may be problematic in practice. We study how much lattice sampling can be sped up using floating-point arithmetic. First, we show that a direct floating-point implementation of these algorithms does not give any asymptotic speedup: the floating-point precision needs to be greater than the security parameter, leading to an overall complexity $\tilde{\mathcal{O}}(n^3)$ where n is the lattice dimension. However, we introduce a laziness technique that can significantly speed up these algorithms. Namely, in certain cases such as NTRUSign lattices, laziness can decrease the complexity to $\tilde{\mathcal{O}}(n^2)$ or even $\tilde{\mathcal{O}}(n)$. Furthermore, our analysis is practical: for typical parameters, most of the floating-point operations only require the double-precision IEEE standard.

1 Introduction

Lattice-based cryptography has been attracting considerable interest in the past few years (see the survey [22]), due to unique features such as security based on worst-case assumptions [3] or more recently fully-homomorphic encryption [11]. But it has several differences compared to classical public-key cryptography based on factoring and discrete logarithms: in particular, the description of many lattice schemes (such as the seminal Ajtai-Dwork cryptosystem [4] and its LWE variants [27], or schemes using lattice sampling [12]) involves real numbers at some point. Although the descriptions usually mention that one can replace these real numbers by approximations with sufficiently high precision, which guarantees efficiency in an asymptotical sense, the practical impact is unclear: no article seems to specify exactly which precision one should take, and how all the operations will be performed exactly. This was not an issue when lattice-based cryptography was considered to be mostly of theoretical interest, but recent works [22,26,17,28,18,20] suggest that the time has come to assess the practicality of lattice-based constructions.

There is another reason to study carefully the use of floating-point arithmetic in lattice-based cryptography. Many recent lattice schemes (*e.g.* trapdoor

signatures [12,6] and ID-based encryption [12,7,1,2]) require a *Gaussian sampler*, that is an efficient algorithm to sample lattice points according to a Gaussian-like distribution, given a (short secret) basis and a target vector. There are two approaches for this task: Klein's randomized variant [15] (as analyzed by Gentry *et al.* [12]) of Babai's nearest plane algorithm [5], and algorithms [26,20] based on convolution for the so-called q-ary lattices.

The cost of Klein's algorithm is the same as Babai's algorithm, namely $\tilde{\mathcal{O}}(n^3 \log B)$ (or $\mathcal{O}(n^4 \log^2 B)$ without fast integer arithmetic), where n is the lattice dimension, and B is the maximal norm of the input basis vectors: since B is polynomial in n for trapdoor bases used in lattice cryptography, the usual cost is $\tilde{\mathcal{O}}(n^3)$ (or $\tilde{\mathcal{O}}(n^4)$ without fast integer arithmetic). The main reason behind the cost of Klein's algorithm is the use of long-integer arithmetic: it relies on Gram-Schmidt orthogonalization, which involves rational numbers of bit-length $\mathcal{O}(n \log B)$. A natural way to improve the efficiency is to use floating-point arithmetic (FPA) to replace exact Gram-Schmidt by suitable approximations. Indeed, Klein's algorithm is a variant of Babai's nearest plane algorithm, which itself is simply the size-reduction subroutine used extensively in the LLL algorithm [16]; and floating-point arithmetic is classically used to speed up LLL (see [29,25,23]). But the use of FPA is not straightforward, and it is unclear at first sight how much speed up can be gained, if any.

On the other hand, the convolution algorithms [26,20] based on Peikert's work [26] have two phases: an offline phase (depending on the secret basis only) and an online phase (depending on the target vector). The online phase costs $\tilde{\mathcal{O}}(n^2)$ for q-ary lattices (which are widespread in lattice cryptography), or even $\tilde{\mathcal{O}}(n)$ in the so-called ring setting (*i.e.* special lattices such as NTRU lattices); but the offline phase is the generation of a noise following some discrete Gaussian distribution, which seems to have the same cost $\tilde{\mathcal{O}}(n^3)$ as Klein's algorithm, and involves floating-point arithmetic whose exact cost is not analyzed in [26,20]. Both algorithms [26,20] can use the same offline phase, which will later be referred to as Peikert's offline Algorithm.

It should be stressed that the offline phase is not a precomputation: this phase must be repeated before each sampling, which is reminiscent of DSA one-time pairs (k, k^{-1}), which can be precomputed as coupons or generated online; but unlike a precomputation it should not be re-used. In some scenario, this computational cost might be acceptable, but it is clearly valuable to analyze and improve the offline phase.

Our results. We develop techniques to improve all three samplers, obtaining the first algorithms with quasi-optimal complexity to sample the discrete Gaussian distribution over lattices: their running time is quasi-linear in the size of the input basis. More precisely, our optimized variant of Klein's algorithm runs in $\tilde{\mathcal{O}}(n^2)$ (for certain bases) and our variant of Peikert's offline algorithm runs in average time $\tilde{\mathcal{O}}(n)$ in some ring setting (where n is the lattice dimension). In both cases, our improvements do not introduce any loss of quality.

To do so, we study how much lattice sampling can be sped up using FPA. As a starting point, we present FPA variants of Klein's algorithm with statistically

close output. Surprisingly, the basic FPA variant has the same asymptotical complexity $\tilde{\mathcal{O}}(n^3)$ as Klein's algorithm, because the precision needs to be greater than the security parameter. However, we also present an optimized algorithm with an improved complexity $\tilde{\mathcal{O}}(n^2)$: it is based on a so-called laziness technique which combines high and low precision FPA. But this optimized complexity only applies to a special class of bases which include NTRUSign bases [13], namely the inverse basis must be small.

Next, we show that the same optimization can be used to speed up Peikert's offline algorithm, improving the total complexity, to bring its offline complexity down to that of its online complexity for both sampling algorithms of [26,20]. More precisely, we apply our laziness technique to reduce the offline complexity to $\tilde{\mathcal{O}}(n^2)$. And for certain ring settings (precisely when the ring is $\mathcal{R} = X^b \pm 1$), we show that the offline phase can also be sped up to average quasi-linear time. This is achieved by using two additional tricks: a structured square-root algorithm and an improved rejection sampler for Gaussians over $\mathbb{Z}$.

As a direct application of this last result, one can strengthen the security of NTRUSign [13] by replacing their heuristic perturbation technique with our optimized sampler, without any loss of efficiency asymptotically. This prevents learning attacks [24,10] on NTRUSign as the signature scheme is now provably secure in the random-oracle model (see [12]), under the (reasonable) assumption that finding close vectors in NTRUSign lattices is hard.

While numerical analysis has often be used [29,25,23] to speed up lattice reduction algorithms in a rigorous way, our work might be its first application to provable security.

Practical impact of laziness. The precision used for floating-point arithmetic has non-negligible practical impact, because fp-operations become much more expensive when the precision goes over the hardware precision. For instance, modern processors typically provide floating-point arithmetic following the double IEEE standard (53-bit precision), but quad-float FPA (113-bit precision simulated by software libraries) is usually about 10-20 times slower for basic operations, and the overhead is much more for multiprecision FPA.

Our complexity results are stated in an asymptotical manner, but our analysis can give concrete bounds (which are provided in the full version [9]). It turns out that in typical cryptographic settings, the *double-precision* (53-bit) IEEE standard can be selected as the "low precision" of our lazy algorithm, which means that most of our fp-operations are hardware fp-operations, even though the security level is not limited to 53 bits.

Roadmap. We start in Sect. 2 with background and notation on lattices, sampling and FPA. In Sect. 3, we present our basic FPA variant of Klein's algorithm, which we optimize using laziness in Sect. 4. In Sect. 5, we apply laziness to speedup Peikert's Offline Algorithm. Eventually, in Sect. 6, we explain how to reach quasi-linear time complexity in the ring setting. Missing proofs and additional details, such as non-asymptotic bounds can be found in the full version [9].

2 Preliminaries

Throughout the paper, we use row representations of matrices (to match lattice software), and use bold fonts to denote vectors: if $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is a matrix, then its row vectors are the $\mathbf{b}_i$'s. Notation $\mathcal{M}_n$ and $\mathcal{S}_n^+$ denote respectively the square matrices, and the square symmetric definite positive matrices of dimension n over $\mathbb{R}$.

2.1 Notation

Lattices. Lattices are discrete subgroups of $\mathbb{R}^m$. A lattice L is represented by a *basis*, that is, a set of linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^m$ such that L is equal to the set $L(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^n x_i \mathbf{b}_i, x_i \in \mathbb{Z}\}$ of all integer linear combinations of the $\mathbf{b}_i$'s. The integer n is the *dimension* of the lattice L. The *volume* $\mathrm{vol}(L)$ is the n-dimensional volume of the parallelepiped generated by any basis of L. In lattice-based cryptography, one mainly uses the so-called q-ary lattices, which include NTRU lattices [14,13] and Ajtai's worst-case/average-case lattices [3]. A *q-ary* lattice is simply a full-rank integer lattice $L \subseteq \mathbb{Z}^n$ such that $q\mathbb{Z}^n \subseteq L$, where q is a somewhat small integer. For such a lattice, $\mathrm{vol}(L)$ divides q^n.

Norms. For a vector $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$ will denote its Euclidean norm. The norm of a matrix B is the maximal norm of its rows: $\|B\| = \max_{i=1}^n \|\mathbf{b}_i\|$. The spectral norm of a square $n \times n$ matrix M is: $\|M\|_s = \max_{\mathbf{x} \in \mathbb{R}^n / \{0\}} \frac{\|\mathbf{x} \cdot M\|}{\|\mathbf{x}\|}$.

Orthogonalization. An $n \times m$ basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ can be written uniquely as $B = \mu \cdot D \cdot Q$ where $\mu = (\mu_{i,j})$ is an $n \times n$ lower-triangular matrix with unit diagonal, D an n-dimensional positive diagonal matrix and Q an $n \times m$ matrix with orthonormal row vectors. Then μD is a lower triangular representation of B (with respect to Q), $B^\star = DQ = (\mathbf{b}_1^\star, \ldots, \mathbf{b}_n^\star)$ is the Gram-Schmidt orthogonalization of the basis, and D is the diagonal matrix formed by the $\|\mathbf{b}_i^\star\|$'s. With those notations, we have $\mu_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^\star \rangle / \|\mathbf{b}_j^\star\|^2$.

For any $\sigma > 0$, we let $\sigma_i = \sigma / \|\mathbf{b}_i^\star\|$ and $\hat{\sigma} = \max_{i=1}^n \sigma_i$. Since the $\mathbf{b}_i^\star$'s are orthogonal, we have $\hat{\sigma} = \sigma / (\min_{i=1}^n \|\mathbf{b}_i^\star\|) = \sigma \left\|B^{\star -1}\right\|_s \leq \sigma \left\|B^{-1}\right\|_s \|\mu\|_s \leq \sigma \left\|B^{-1}\right\|_s n\hat{\mu}$ where $\hat{\mu} \geq 1$ upper bounds the coefficients of μ.

Gaussian Distribution. The (unnormalized) weight of Gaussian distribution of parameter $\sigma \in \mathbb{R}$ and center $c \in \mathbb{R}$ at $x \in \mathbb{R}$ is defined by $\rho_{\sigma,c}(x) = \exp\big(-\pi \frac{(x-c)^2}{\sigma^2}\big)$, and more generally by $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = \exp\big(-\pi \frac{\|\mathbf{x}-\mathbf{c}\|^2}{\sigma^2}\big)$ for $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$. The discrete Gaussian distribution over $\mathbb{Z}$ is defined by $D_{\mathbb{Z},\sigma,c}(x) = \rho_{\sigma,c}(x)/\rho_{\sigma,c}(\mathbb{Z})$, and more generally, over a lattice L by $D_{L,\sigma,\mathbf{c}}(\mathbf{x}) = \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\rho_{\sigma,\mathbf{c}}(L)$. Peikert [26] generalized the discrete Gaussian distribution over a lattice L using a positive definite matrix $\Sigma > 0$ (which generalizes $\sigma \in \mathbb{R}$) as follows: the density $D_{L,\sqrt{\Sigma},\mathbf{c}}(\mathbf{x})$ is proportional to $\rho_{1,0}((\mathbf{x}-\mathbf{c})B^{-1})$ where $\Sigma = B^t B$, for $\mathbf{x} \in L$.

2.2 Gaussian Lattice Sampling

The goal of Gaussian lattice sampling is to efficiently sample lattice points according to a distribution statistically close to $D_{L,\sigma,\mathbf{c}}$. All lattice samplers known [15,12,26,20] have constraints on the parameter σ and the statistical distance, which are related to the so-called smoothing parameter. The sampling parameter σ determines the average distance of the sampled lattice point to the target point: the smaller σ, the better for cryptographic applications. For instance, σ impacts the verification threshold of lattice-based signatures [12] and therefore the security of the scheme; a lower quality forces to increase lattice parameters. And for a security level of λ bits, we need a statistical distance less than $2^{-\lambda}$.

Smoothing Parameter. For any n-dimensional lattice L and any real $\iota > 0$, the *smoothing parameter* $\eta_\iota(L)$ (see [21]) is the smallest real $s > 0$ such that $\rho_{1/s}(L^* \setminus \{\mathbf{0}\}) \leq \iota$, where L^* is the dual lattice of L. For details on the importance of this parameter, please refer to [21,12].

Klein's sampling. Gentry *et al.* showed in [12] that given as input a lattice basis B of an n-dimensional lattice L such that $\sigma \geq \|B^*\|\omega(\sqrt{\log n})$, Klein's algorithm [15] outputs lattice points with a distribution statistically close to $D_{L,\sigma,\mathbf{c}}(\mathbf{x})$. For applications, it is more convenient to have a concrete bound on the statistical distance, and to separate this bound from the lattice dimension n. We therefore use the following concrete analysis of Klein's algorithm:

Theorem 1 (Concrete version of [12, Th. 4.1]). *Let $n, \lambda \in \mathbb{N}$ be any positive integers, and $\iota = 2^{-\lambda}/(2n)$. For any n-dimensional lattice L generated by a basis $B \in \mathbb{Z}^{n \times n}$, and for any target vector $\mathbf{c} \in \mathbb{Z}^{1 \times n}$, Alg. 2 is such that the statistical distance $\Delta(D_{L,\sigma,\mathbf{c}}, \mathbf{SampleLattice}_\infty(B, \sigma, \mathbf{c}))$ is less than $2^{-\lambda}$, under the condition:*

$$\sigma \geq \|B^\star\|\, \eta_\iota(\mathbb{Z}) \qquad \text{where } \eta_\iota(\mathbb{Z}) \lessapprox \sqrt{(\lambda \ln 2 + \ln n)/\pi}\ .$$

Tailcut. We will also use a tailcut parameter τ, chosen such that (informally) a sample from a normal distribution of parameter σ is at distance at most $\tau\sigma$ from the center with overwhelming probability:

Corollary 1 (Tailcut error, Corollary of [21, Lemma 2.10]). *Let L be an n-dimensional lattice, $\iota \leq 1/2$, $\sigma \geq \eta_\iota(L)$, $\tau > 1$ $\delta_\tau \in (0,1)$ and $\mathbf{c} \in \mathbb{R}^n$. For $x \leftarrow D_{L,\sigma,\mathbf{c}}$ we have: $\Pr\left[\|\mathbf{x} - \mathbf{c}\| \geq (1-\delta_\tau)\tau\sigma\right] \leq 3E_{\mathbf{tailcut}}(\tau, \delta_\tau)^n$ where $E_{\mathbf{tailcut}}(\tau, \delta_\tau) \stackrel{def}{=} \tau\sqrt{2\pi e} \cdot e^{-\pi(1-\delta_\tau)^2\tau^2}$.*

2.3 Floating-point arithmetic

We consider floating-point arithmetic (FPA) with m bits of mantissa, which we denote by $\mathbb{FP}_m$: the precision is $\epsilon = 2^{-m+1}$. A floating-point number $\bar{f} \in \mathbb{FP}_m$ is a triplet $\bar{f} = (s, e, v)$ where $s \in \{0,1\}$, $e \in \mathbb{Z}$ and $v \in \mathbb{N}_{2^m-1}$, which represents the

real number $R(\bar{f}) = (-1)^s \cdot 2^{e-m} \cdot v \in \mathbb{R}$. Every FPA-operation $\bar{\circ} \in \{\bar{+}, \bar{-}, \bar{\times}, \bar{/}\}$ and its respective arithmetic operation on $\mathbb{R}$, $\circ \in \{+, -, \cdot, /\}$ verify:

$$\forall \bar{f}_1, \bar{f}_2 \in \mathbb{FP}_m, \left|R(\bar{f}_1 \bar{\circ} \bar{f}_2) - (R(\bar{f}_1) \circ R(\bar{f}_2))\right| \leq (R(\bar{f}_1) \circ R(\bar{f}_2))\epsilon \tag{1}$$

We require a floating-point implementation of the exponentiation function $\bar{\exp}(\cdot)$ and we assume that it verifies a similar error bound: for any $\bar{f} \in \mathbb{FP}_m$, $\left|R(\bar{\exp}(\bar{f})) - \exp(R(\bar{f}))\right| \leq \epsilon$. Finally, we note that if an integer $x \in \mathbb{Z}$ verifies $|x| \leq 2^m$, it can be converted to a float $\bar{f} \in \mathbb{FP}_m$ with no error, i.e. $\mathbb{R}(\bar{f}) = x$. For the rest of the article, we omit the function R and consider $\mathbb{FP}_m$ as a subset of $\mathbb{R}$.

2.4 Pseudo-code

Types. Variables are typed, and the type is given at each initialization and assignment, as follows: *variable* $\leftarrow$ *value* : *type*. We use a simpler syntax for the definition of local functions: $\{variable \mapsto value\}$. Functional types are denoted by $(t_1 \rightarrow t_2)$.

Primitives. We use the basic arithmetic operations $\{+, -, \cdot, /\}$, as well as squaring $\square^2$ and exponentiation exp; the arguments are either integers in $\mathbb{Z}$, or floating-point numbers in $\mathbb{FP}_m$. We extend these notations to vectors and matrices. We also use the following additional primitives:
$\mathbf{RandInt}(a, b) : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$: return a random uniform integer in the range $[a, b]$.
$\mathbf{RandFloat}_m() : \text{void} \rightarrow \mathbb{FP}_m$: return a random uniform float in the range $[0, 1)$.
$\mathbf{ExtRandFloat}_{m',m}(r) : \mathbb{FP}_{m'} \rightarrow \mathbb{FP}_m$: return a random uniform floating-point number in the range $[r, r+2^{-m'})$. For a random $r \leftarrow \mathbf{RandFloat}_{m'}()$, the output follows the same distribution as $\mathbf{RandFloat}_m()$.

3 A Basic Floating-Point Variant of Klein's Algorithm

3.1 Description

Algorithm 2 describes both Klein's algorithm [15] and our basic floating-point variant: given a basis B of a lattice L, a target $\mathbf{c}$ and a parameter σ, the algorithm outputs a vector with distribution statistically close to $D_{L,\sigma,\mathbf{c}}$. It uses two subroutines: $\mathbf{DecomposeGS}_m$ (Alg. 3) to compute the coordinates t_i's of the target vector $\mathbf{c}$ with respect to the Gram-Schmidt basis $B^\star$, and $\mathbf{Sample}\mathbb{Z}_m$ (Alg. 1) to sample according to the Gaussian distribution over $\mathbb{Z}$. Algorithm 2 comes in two flavors:

- $\mathbf{SampleLattice}_\infty$ is the exact version, which corresponds to Klein's original algorithm [15]. The $\mu_{i,j}$'s and the t_i's are represented exactly by rational numbers, and all the computations use exact integer arithmetic. Assuming $\sigma \in \mathbb{Q}$, we can only ensure that $\sigma_i \in \sqrt{\mathbb{Q}}$, thus we can represent them exactly by their square. We also assume that this version has access to a perfect

primitive (or an oracle) $\mathbf{Sample}\mathbb{Z}_\infty(\sigma_i, t_i, \tau = \infty)$ that given $t_i, \sigma_i^2 \in \mathbb{Q}$ answers an integer $x : \mathbb{Z}$ exactly according to the distribution $D_{\mathbb{Z},\sigma_i,t_i}$. It does not matter how to sample such a perfect distribution, as the purpose of this perfect algorithm is to be a reference for inexact ones.

– $\mathbf{SampleLattice}_m$ is our basic floating-point version, using $\mathbb{FP}_m$. The matrices μ and $B^\star$ and values σ_i may have been pre-computed exactly, but only approximations are stored.

Algorithm 1 $\mathbf{Sample}\mathbb{Z}_m$: Rejection Sampling for Discrete Gaussian on $\mathbb{Z}$

input: A center $t : \mathbb{FP}_m$, and a parameter $\sigma : \mathbb{FP}_m$, and a tailcut parameter $\tau : \mathbb{FP}_m$
output: output $x : \mathbb{Z}$, with distribution statistically close to $D_{\mathbb{Z},t,\sigma}$

1: $h \leftarrow -\pi/\sigma^2 : \mathbb{FP}_m$; $x_{\max} \leftarrow \lceil t + \tau\sigma \rceil : \mathbb{Z}$; $x_{\min} \leftarrow \lfloor t - \tau\sigma \rfloor : \mathbb{Z}$
2: $x \leftarrow \mathbf{RandInt}(x_{\min}, x_{\max}) : \mathbb{Z}$; $p \leftarrow \exp(h \cdot (x - t)^2) : \mathbb{FP}_m$
3: $r \leftarrow \mathbf{RandFloat}_m() : \mathbb{FP}_m$; **if** $r < p$ **then** return x
4: **Goto** Step 2.

Algorithm 2 $\mathbf{SampleLattice}_m$: Gaussian Sampling over a lattice

input: a (short) lattice basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n) : \mathbb{Z}^{n\times n}$, parameter $\sigma : \mathbb{FP}_m$, A target vector $\mathbf{c} : \mathbb{Z}^{1\times n}$, and a tailcut parameter $\tau : \mathbb{FP}_m$ **Precomputation:** The GS decomposition $(B^\star = (\mathbf{b}_1^\star, \ldots, \mathbf{b}_n^\star), (\mu_{i,j}) = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_n))$, norms $r_i = \|\mathbf{b}_i^\star\| : \mathbb{FP}_m$ and $\sigma_i = \sigma/r_i : \mathbb{FP}_m$
output: a vector $\mathbf{v} : \mathbb{Z}^{1\times n}$ drawn approximately from $D_{L,c,\sigma}$ where $L = L(B)$

1: $\mathbf{v}, \mathbf{z} \leftarrow \mathbf{0} : \mathbb{Z}^n$; $\mathbf{t} \leftarrow \mathbf{DecomposeGS}_m(\mathbf{c}, B^\star) : \mathbb{FP}_m$
2: **for** $i = n$ downto 1 **do**
3: $z_i \leftarrow \mathbf{Sample}\mathbb{Z}_m(\sigma_i, t_i, \tau) : \mathbb{Z}$
4: $\mathbf{v} \leftarrow \mathbf{v} + z_i \cdot \mathbf{b}_i : \mathbb{Z}^n$; $\mathbf{t} \leftarrow \mathbf{t} - z_i \cdot \boldsymbol{\mu}_i : \mathbb{FP}_m^n$
5: **end for**
6: return $\mathbf{v}$

Algorithm 3 $\mathbf{DecomposeGS}_m$: Decompose a vector c over the GS Basis

input: A vector $\mathbf{c} : \mathbb{Z}^{1\times n}$, an orthogonal basis $B^\star = (\mathbf{b}_1^\star, \ldots, \mathbf{b}_n^\star) : \mathbb{Q}^{n\times n}$, and $r_i^2 = \|\mathbf{b}_i^\star\|^2 \in \mathbb{FP}_m$
output: output $\mathbf{t} : \mathbb{Q}^n$ such that $c = t_1\mathbf{b}_1^\star + \cdots + t_n\mathbf{b}_n^\star$

1: $\mathbf{y} \leftarrow \mathbf{c} \cdot B^{\star t}$ $: \mathbb{Z}^{1\times n}$
2: return $(y_1/r_1^2, \ldots, y_n/r_n^2)$

The description of $\mathbf{SampleLattice}_\infty$ differs from the original description [15,12] only in the way we compute and update the coordinates t_i's. In our version, the final value of t_i before it is used is $t_i = \langle \mathbf{c}, \mathbf{b}_i^\star \rangle / r_i^2 - \sum_{j>i}^n z_j \mu_{j,i}$, which matches with the original value :

$$t_i' = \left\langle \mathbf{c} - \sum_{j>i}^n z_j \mathbf{b}_j \ , \ \mathbf{b}_i^\star \right\rangle / r_i^2 = \left(\langle \mathbf{c}, \mathbf{b}_i^\star \rangle - \sum_{j>i}^n z_j \langle \mathbf{b}_j, \mathbf{b}_i^\star \rangle \right) / r_i^2 = t_i$$

We unroll this computation and update the sum after each value z_i is known. This allows a parallelization up to n processors without the usual $\log n$ factor required for summing up all terms.

Since we use the matrix μ in the main loop, we might want to get rid of $B^\star$ for the **DecomposeGS** algorithm, to save some precomputation and storage, by computing $\mathbf{c}' \leftarrow \mathbf{c} \cdot B^t$ and then solving the triangular system $\mathbf{y}\ \mu^t = \mathbf{c}'$. Solving this system also requires n^2 operations, however when using FPA, it would produce a relative error exponential in the dimension n, because we recursively use previous results.

Our main loop may also be seen as solving a triangular system, where we apply Gaussian rounding at each step. It is worth noting that this additional rounding prevents such relative exponential error, as our proof will show.

Efficiency of $\mathbf{SampleLattice}_\infty$. The algorithm $\mathbf{SampleLattice}_\infty$ performs $\mathcal{O}(n^2)$ arithmetic operations on rational numbers of size $\mathcal{O}(n \log B)$, which leads to a complexity of $\tilde{\mathcal{O}}(n^4)$ for cryptographic use. Here, we ignored the calls to the oracle $\mathbf{Sample}\mathbb{Z}_\infty(\cdot, \cdot, \tau = \infty)$.

Termination of $\mathbf{Sample}\mathbb{Z}_\infty(\cdot, \cdot, \tau < \infty)$. We upper bound the number of trials of Rejection Sampling, ignoring issues related to the transcendental function exp:

Fact 2 *If $\sigma \geq 4$ and $\tau \geq 1$, and uniforms $x \leftarrow \mathbb{Z} \cap [x_{min}, x_{max}]$ and $r \leftarrow [0, 1)$, we have $\Pr\left[r < \rho_{\sigma,t}(x)\right] > 1/(6\tau)$ where $x_{min} = \lceil t - \tau\sigma \rceil$ and $x_{max} = \lfloor t + \tau\sigma \rfloor$.*

Thus $\mathbf{Sample}\mathbb{Z}_\infty(\cdot, \cdot, \tau)$ performs less than 6τ trials on average.

3.2 Correctness

We give the list of assumptions needed for our correctness results (Theorems 3 and 5), and which we refer to as conditions $\mathcal{A}$.

Assumption on Gram-Schmidt precomputation. We assume that the Gram-Schmidt values are (possibly approximately) precomputed, and that the computed values $\bar{\mu}_{i,j}$, $\bar{b}^\star_{i,j}$ and $\bar{\sigma}_i$ verify:

$$|\Delta\mu_{i,j}| = |\mu_{i,j} - \bar{\mu}_{i,j}| \leq \hat{\mu}\epsilon,\ \left|\Delta b^\star_{i,j}\right| = \left|b^\star_{i,j} - \bar{b}^\star_{i,j}\right| \leq \|\mathbf{b}^\star_i\|\,\epsilon,$$
$$|\Delta\sigma_i| = |\sigma_i - \bar{\sigma}_i| \leq \sigma_i\epsilon,$$

where $\hat{\mu}$ denotes the maximal absolute value of the sub-diagonal coefficient of μ. Those condition can be achieved by running the precomputation exactly, then convert the result to floating points of mantissa size m.

Assumption on the target vector. We assume that the components c_i of the input target vector $\mathbf{c}$ satisfy: $|c_i| \leq q$ for a parameter q. This holds in all known cryptographic applications of lattice sampling, for which the lattice is q-ary. But we do not require that the lattice is q-ary.

Assumption on the parameters.

$$\mathcal{A}\begin{cases}\epsilon \le 0.01, & K^n = (1+\epsilon)^n \le 1.1,\ 1 + nK^n\epsilon \le 1.01\\ n\iota \le 0.01, \forall i, \sigma_i \ge \eta_\iota(\mathbb{Z}), & \forall i, \sigma_i \ge 4\\ n \ge 10 \quad \tau \ge 4\end{cases}$$

The assumptions on ϵ are easily achievable for a mantissa size m at least logarithmic in the dimension n. The condition on ι is not restrictive as it needs to be negligible. Similarly, conditions on σ_i's are not restrictive since the security requires all $\sigma_i \ge \eta_\iota(\mathbb{Z}) > 4$ for security parameters $\lambda \ge 80$.

For the rest of the analysis, we assume that all parameters B, $\mathbf{c}$ and σ are fixed. Our main result states that with enough precision, the outputs of the exact sampler $\mathbf{SampleLattice}_\infty$ and the floating-point sampler $\mathbf{SampleLattice}_m$ are statistically close:

Theorem 3. *There exist constants C_λ, C_τ, C_m, such that for any security parameter $\lambda \ge C_\lambda$, and under conditions $\mathcal{A}$, the statistical distance between $\mathbf{SampleLattice}_m$ and $\mathbf{SampleLattice}_\infty$ is less than $2^{-\lambda}$ on the same input if the following conditions are satisfied:*

$$\tau \ge C_\tau\sqrt{\lambda \log n} \quad m \ge C_m + \lambda + 2\log_2(\left\|B^{-1}\right\|_s) + \log_2\left(\hat{\mu}^2 n^4 (q+\sigma^2)\tau^3\right)$$

Furthermore, under those conditions, the integers manipulated by $\mathbf{SampleLattice}_m$ can be represented by floating-point numbers without errors.

3.3 Efficiency

We deduce the efficiency of the basic floating-point sampler from Theorem 3. We first analyze $\mathbf{Sample}\mathbb{Z}_m$:

Fact 4 *There is a constant C_m such that for any $m \ge C_m$, and any $\tau \ge 1$, $\mathbf{Sample}\mathbb{Z}_m(\cdot,\cdot,\tau)$ performs less than 6τ trials on the average.*

This can be easily derived from Fact 2 and appropriate error bound (see full version). This ensures that $\mathbf{SampleLattice}_m$ performs $\sim 6n^2$ $\mathbb{FP}_m$-operations as long as $\tau = o(n)$.

Arbitrary bases. To minimize the FPA-precision m in Theorem 3, we need to evaluate $\log(\left\|B^{-1}\right\|_s)$: this is always less than $\approx n\log(B)$ by Cramer's rule. This leads to the constraint $m \ge \lambda + n\ell$ where ℓ is logarithmic in n and B, yielding a $\tilde{\mathcal{O}}(n^3)$ bit-complexity as long as $\lambda = \mathcal{O}(n)$, or $\tilde{\mathcal{O}}(n^4)$ without fast integer arithmetic.

The exact algorithm $\mathbf{SampleLattice}_\infty$ also has complexity $\tilde{\mathcal{O}}(n^3)$. However, the constants are likely to be smaller for the FPA sampler. Indeed, the exact algorithm must handle integers of size $\log(\max_{1\le i\le n}\mathrm{vol}(\mathbf{b}_1,\ldots,\mathbf{b}_i))$, whereas the quantity $\log(\left\|B^{-1}\right\|_s)$ is typically smaller, though they have similar worst-case asymptotical bounds. And the constants of the FPA sampler can be improved by processing the basis, for instance using LLL reduction.

Furthermore, in cryptographic applications, we may focus on bases B of a particular shape. More precisely, we will consider the following type of basis:

Small-inverse bases. A sequence $\mathcal{C} = (\mathcal{C}_n)$ of square matrices generating q_n-ary lattices of dimension n is a class of *small-inverse bases* if there exists a polynomial function f such that for any basis $B \in \mathcal{C}_n$, $\|B\|_s \leq f(n)$ and $\left\|B^{-1}\right\|_s \leq f(n)$.

In particular, the bases used by the NTRUSign signature scheme [13] form a small-inverse class (see [13]). For such bases, we only need $m \geq \lambda + \ell$ for ℓ logarithmic in λ. This still gives a $\tilde{\mathcal{O}}(n^3)$ complexity for cryptographic use (when $\lambda \sim n$), but with much better constants.

4 A Lazy Floating-Point Variant of Klein's Algorithm

Overview. We now describe our optimized sampler, which is more efficient than the basic sampler, due to a better use of FPA. The analysis of the basic sampler showed that it was sufficient to compute t_i up to $\approx \lambda$ bits below the unity to get an error below $2^{-\lambda}$ on the output distribution. However, a careful analysis of the rejection sampling algorithm (Alg. 1) shows that most of the time, many of those bits are not used: the precision of t_i impacts the precision of $p = \rho_{\sigma,t}(x)$, which is only used to make a comparison with a uniform random real $r \in [0, 1)$. For all $j > 1$, such a comparison is determined by the first j bits, except with probability 2^{-j} (exactly when the j first bits of r and p match); and on average only the first two bits contribute to the decision.

However, we still need to decide properly this comparison even when the first $j \leq \lambda$ bits match, to output a proper distribution. This suggests a new strategy: compute lazily the bits of t_i and p. We first only compute most significant bits and backtrack for additional bits until the comparison can be determined. We choose a simple lazyness control, using only two levels of precision (for simplicity, but also for practical efficiency). Informally, we choose $k \leq \lambda$, and compute t_i up to a precision m' that only guarantees the first k bits of p, draw the first k bits of the random real r. If the comparison is decided with those k bits, continue normally. Otherwise (which happens with probability less than 2^{-k}), recompute t_i and p at a precision m to ensure λ correct bits.

4.1 Description

Our optimized sampler $\mathbf{LazySampleLattice}_{m',m}$ (Alg 4) works with two floating-point types, $\mathbb{FP}_m$ (high precision) and $\mathbb{FP}_{m'}$ (low precision), where $m > m'$. The algorithm works similarly to the original one, except it now works most of the time at low precision m'. The subroutine for sampling over $\mathbb{Z}$ is replaced by $\mathbf{LazySample}\mathbb{Z}_{m',m}$, which takes the usual arguments at low precision, plus an error bound, and access to high-precision arguments: σ is precomputed thus requiring no special care, however, the access to high precision value of t is given through a function that takes no argument.

This new subroutine $\mathbf{LazySample}\mathbb{Z}_{m',m}$ (Alg. 5) works identically to the original $\mathbf{Sample}\mathbb{Z}_{m'}$ as long as the decisive comparison is trusted, i.e. as long as the difference $|r' - p'|$ is higher than the error bound δ_p. Otherwise, the high precision is triggered, and high-precision inputs are requested through the function F. Then all sample trials are computed with high precision.

Algorithm 4 $\mathbf{LazySampleLattice}_{m',m}$: Lazy Gaussian Sampling over a lattice

input: Same as **SampleLattice** plus low precision versions of $\mu, B^\star$ and σ_i's values: $\mu', B^{\star\prime} : \mathbb{FP}_{m'}^{n\times n}$, $\sigma'_i : \mathbb{FP}_{m'}$, and an error bound δ_p

output: Same as **SampleLattice**

1: $\mathbf{v}, \mathbf{z} \leftarrow \mathbf{0} : \mathbb{Z}^n$; $\quad \mathbf{t}' \leftarrow \mathbf{DecomposeGS}_{m'}(\mathbf{c}, B^{\star\prime}) : \mathbb{FP}_{m'}^n$
2: **for** $i = n$ downto 1 **do**
3: $\quad F_i \leftarrow \{() \mapsto \langle \mathbf{c}, \mathbf{b}_i^\star \rangle - \langle \mathbf{z}, \lceil \mu^t \rfloor_i \rangle\}$: (void $\to \mathbb{FP}_m$)
4: $\quad z_i \leftarrow \mathbf{LazySample}\mathbb{Z}_{m',m}(\sigma'_i, \tau, t'_i, \delta_p, \sigma_i, F_i) : \mathbb{Z}$
5: $\quad \mathbf{v} \leftarrow \mathbf{v} + z_i \cdot \mathbf{b}_i : \mathbb{Z}^n$; $\quad \mathbf{t}' \leftarrow \mathbf{t}' - z_i \cdot \boldsymbol{\mu}'_i : \mathbb{FP}_{m'}^n$
6: **end for**
7: return $\mathbf{v}$

Algorithm 5 $\mathbf{LazySample}\mathbb{Z}_{m',m}(\sigma', \tau, t', \delta_p : \mathbb{FP}_{m'},\ \sigma : \mathbb{FP}_m, F : (\text{void} \to \mathbb{FP}_m))$

1: $h' \leftarrow -\pi/\sigma^2 : \mathbb{FP}_{m'}$; $x_{\max} \leftarrow \lceil t' + \tau\sigma' \rceil : \mathbb{Z}$; $x_{\min} \leftarrow \lfloor t' - \tau\sigma' \rfloor : \mathbb{Z}$; highprec $\leftarrow$ false : bool
2: $x \leftarrow \mathbf{RandInt}(x_{\min}, x_{\max}) : \mathbb{Z}$; $r' \leftarrow \mathbf{RandFloat}_{m'}() : \mathbb{FP}_{m'}$
3: **if** not(highprec) **then**
4: $\quad p' \leftarrow \exp(h' \cdot (x - t')^2) : \mathbb{FP}_{m'}$
5: $\quad$ **if** $|r' - p'| \le \delta_p$ **then** $\{t \leftarrow F() : \mathbb{FP}_m$; $h \leftarrow -\pi/\sigma^2 : \mathbb{FP}_m$; highprec $\leftarrow$ true $\}$
6: $\quad$ **else if** $r' < p'$ **then** return x
7: **end if**
8: **if** highprec **then**
9: $\quad r \leftarrow \mathbf{ExtRandFloat}_{m',m}(r') : \mathbb{FP}_m$; $\quad p \leftarrow \exp(h \cdot (x - t)^2) : \mathbb{FP}_m$
10: $\quad$ **if** $r < p$ **then** return x
11: **end if**
12: **Goto** Step 2.

4.2 Correctness

We need to determine a proper value for the error bound δ_p in terms of the basis and m' (the size of the low precision), to ensure correctness. For this parameter, the lower the better, since it determines the probability to trigger the re-computation of t at high precision, as detailed in the next section. The behavior of the new subroutine is analyzed by the following:

Lemma 1 (Informal, see [9] for a formal statement). *The behaviour of* $\mathbf{LazySample}\mathbb{Z}_{m,m'}$ *given approximate inputs* $\sigma \pm \delta_\sigma$ *and* $t \pm \delta_t$ *and* δ_p*, is similar to* $\mathbf{Sample}\mathbb{Z}_m$ *on input* σ, t *under the condition:*

$$\delta_p \ge 4\sigma^2\epsilon' + 1.7\sigma\delta_\sigma + (1.7/\sigma)\delta_t \quad \textit{where } \epsilon' = 2^{1-m'}$$

From this lemma, we prove the correctness of $\mathbf{LazySampleLattice}_{m',m}$, summarized by the following result.

Theorem 5. *There exist constants* $C_\lambda, C_\tau, C_m, C'_m, C_{\delta_p}$*, such that for any security parameter* $\lambda \ge C_\lambda$*, and under Conditions* $\mathcal{A}$*, the statistical distance between* $\mathbf{LazySampleLattice}_{m,m'}$ *and* $\mathbf{SampleLattice}_\infty$ *is less than* $2^{-\lambda}$ *on the same*

input if the following conditions are satisfied:

$$\begin{aligned}
\tau &\geq C_\tau \sqrt{\lambda \log n} \\
m &\geq C_m + \lambda + 2\log_2(\left\|B^{-1}\right\|_s) + \log_2\left(\hat{\mu}^2 n^4 q \sigma^2 \tau^3\right) \\
m' &\geq C_{m'} + 2\log_2(\left\|B^{-1}\right\|_s) + \log_2\left(\hat{\mu}^2 n^4 (Q+\sigma^2) \tau^3\right) \\
\delta_p &\geq 2^{-k} \text{ where } k = m' - \left(C_{\delta_p} + 2\log_2(\left\|B^{-1}\right\|_s) + \log_2\left(\hat{\mu}^2 n^3 \tau \sigma^2 q\right)\right)
\end{aligned}$$

Furthermore, under those conditions, the integers manipulated by the algorithm can be represented by low-precision floating-point numbers ($\mathbb{FP}_{m'}$) without errors.

4.3 Efficiency

The error bound δ_p impacts the efficiency of the optimized sampler as follows:

Lemma 2. *Under the conditions of Theorem 5, each call to* $\mathbf{LazySample}\mathbb{Z}_{m,m'}$ *triggers high precision with probability less than* $12\tau\delta_p$*. On the average, the algorithm* $\mathbf{LazySampleLattice}_{m,m'}$ *performs less than* $\mathcal{O}(n^2\tau\delta_p)$ *high-precision floating-point operations.*

Proof. At each trial performed by $\mathbf{LazySample}\mathbb{Z}_{m,m'}$, the probability to trigger high precision is less than $2\delta_p$: indeed it happens only if the randomness $r' \leftarrow [0,1)$ falls in the interval $[p'-\delta_p, p'+\delta_p]$. It remains to bound the average number of trials performed by $\mathbf{LazySample}\mathbb{Z}_{m,m'}$. The condition of Theorem 5 ensures that it behaves similarly to $\mathbf{Sample}\mathbb{Z}_m$. Thus, for a large enough m, Fact 4 ensures that the average number of trials is less than 6τ.

Triggering high precision during $\mathbf{LazySample}\mathbb{Z}_{m,m'}$ requires $\mathcal{O}(n)$ high-precision FPA operations. This subroutine is called n times, thus on the average less than $\mathcal{O}(n^2\tau\delta_p)$ high-precision FPA operations. □

This leads to our main result: with Small-Inverse bases, the discrete Gaussian distribution can be sampled in quasi-quadratic time, with an exponentially small statistical distance, and no sacrifice on the quality compared to the analysis of [12].

Theorem 6 (Gaussian sampling in quasi-quadratic time). *Let* $(\mathcal{C}_n)$ *be a Small-Inverse class of bases. For any implicit function* λ*, such that* $\lambda \sim n$*, and* σ *polynomial in* n*, there exist implicit functions* m, m', τ, δ_p *of* n *such that, for any basis* $B \in \mathcal{C}_n$ *generating a lattice* L*:*

- $\mathbf{LazySampleLattice}_{m,m'}(B, \sigma, \mathbf{c}, \tau, \delta_p)$ *runs in expected time* $\tilde{\mathcal{O}}(n^2)$ *without fast integer arithmetic.*
- $\Delta(D_{L,\sigma,\mathbf{c}}, \mathbf{LazySampleLattice}_{m,m'}(B, \sigma, \mathbf{c}, \tau, \delta_p)) \leq 2^{-\lambda}$ *whenever* σ *verifies* $\sigma \geq \|B^\star\| \, \eta_\iota(\mathbb{Z})$ *with* $\iota = 2^{-\lambda}/(4n)$*.*

Proof. For a small-inverse class of bases, the conditions of Theorem 5 can be satisfied with functions verifying:

$$\tau = \mathcal{O}(\sqrt{n}), m = \mathcal{O}(n), m' = \mathcal{O}(\log n), \delta_p = \mathcal{O}(1/n^{5/2}).$$

Lemma 2 states that on the average, less than $\mathcal{O}(n^2\tau\delta_p)$ high-precision operations are performed, which in our case is a $\mathcal{O}(1)$. Without fast integer arithmetic, the total complexity is thus less than $\mathcal{O}(n^2)\mathcal{O}(m'^2) + \mathcal{O}(1)\mathcal{O}(m^2) \leq \tilde{\mathcal{O}}(n^2)$. □

5 Speeding Up Peikert's Offline Algorithm

Peikert [26] recently proposed a different sampling algorithm based on convolution, which was inspired by NTRUSign's perturbation countermeasure [13]. This algorithm offers a different trade-off than Klein's algorithm, with slightly worse constraints on sampling parameters (see [26] for details). The discrete Gaussian distribution is obtained by adding two points, one generated by an offline phase, the other generated by a (cheaper) online phase. The online phase is essentially a randomized variant of Babai's round-off algorithm [5], which only involves small-integer arithmetic when the input is a q-ary lattice, and thus runs in $\tilde{\mathcal{O}}(n^2)$ time, and even $\tilde{\mathcal{O}}(n)$ in ring settings. This offline phase is itself essentially the generation of some discrete Gaussian distribution, which requires long-integer arithmetic, and is not fully analyzed in [26], but seems to be $\tilde{\mathcal{O}}(n^3)$ (even $\tilde{\mathcal{O}}(n^4)$ without fast integer arithmetic) like Klein's algorithm. In the follow-up work of Micciancio and Peikert [20], a new kind of lattice trapdoor is introduced to optimize efficiency and geometric quality, which allows an even faster online phase, but the same kind of offline computations is required. We refer to this common offline phase as Peikert's offline algorithm.

5.1 Peikert's Offline Algorithm

Let B be the input basis of the lattice for which one wants to generate the discrete Gaussian distribution. In both [26,20], the offline phase consists of generating a (centered) discrete Gaussian noise over $\mathbb{Z}^n$ of parameter $\Sigma \in \mathcal{S}_n^+$ such that $B^tB + \Sigma = sI_n$ where s is some appropriate real number: this implies certain constraints on B which are discussed in [26]. Letting $\Sigma = C^tC$, this distribution $D_{\mathbb{Z}^n,\sqrt{\Sigma}}$ has support $\mathbb{Z}^n$ and density at $\mathbf{x}$ proportional to $\rho_{1,0}(\mathbf{x}C^{-1})$: in other words, this is "essentially' the discrete Gaussian distribution $\mathcal{D}$ over the lattice spanned by C^{-1}, since the density of $\mathbf{x} \in \mathbb{Z}^n$ is proportional to the density of the lattice point $\mathbf{x}C^{-1}$ in $\mathcal{D}$. The offline-phase algorithm is described in Alg. 6 (from [26]): it generates this discrete Gaussian distribution by convolution (see [26]), which is a different strategy than Klein's algorithm, and has different constraints. The main idea is to consider a "shift" $\Sigma' = \Sigma - \eta^2 I_n$ of Σ such that $\Sigma' \in \mathcal{S}_n^+$ (which implies that $\Sigma \geq \eta^2 I_n$) and $\eta \geq \eta_\iota(\mathbb{Z}^n)$, and to compute a square-root L of Σ', *i.e.* $\Sigma' = L^tL$. To implement this, it is suggested in [26] to use a Cholesky decomposition. The parameters selected to reach security λ are $\eta = \tau = \eta_\iota(\mathbb{Z}) = \tilde{\mathcal{O}}(\sqrt{\lambda})$. The choice of the floating-point precision is not discussed in [26,20], however a quick analysis shows that one should take $m = \lambda + \ell$ where ℓ is logarithmic in n, s and τ. Thus, a naive implementation would have a running-time of $\tilde{\mathcal{O}}(n^2\lambda^2)$, the main cost being a (non-structured) matrix-vector product: that is n^2 floating-point operations, at precision $\tilde{\mathcal{O}}(\lambda)$.

Algorithm 6 Peikert's Offline Algorithm

input: $\Sigma \in \mathcal{S}_n^+$, a real $\eta \geq \eta_\iota(\mathbb{Z}^n)$ such that $\Sigma' = \Sigma - \eta^2 I_n \in \mathcal{S}_n^+$ and ι is negligible, and a square-root L of Σ' *i.e.* $\Sigma' = L^t L$.

output: An integer vector $\mathbf{z} \in \mathbb{Z}^n$ following the distribution $D_{\mathbb{Z}^n,\sqrt{\Sigma}}$

1: Choose $\mathbf{x} : \mathbb{R}^n$ according to the continuous Gaussian distribution of covariance I_n
2: $\mathbf{y} = \mathbf{x} \cdot L$
3: **for** $i = 1$ to n **do** $z_i \leftarrow$ **Sample**$\mathbb{Z}_m(\eta, y_i, \tau)$
4: return $\mathbf{z}$

5.2 Using Laziness in Peikert's Offline Algorithm

Like in Klein's sampling algorithm, the offline phase of Peikert's algorithm [26] only uses non-integer values to compute the input of the **Sample**$\mathbb{Z}_m(\eta, \cdot, \tau)$ subroutine. High-precision bits of this input are useless except with small probability: one may apply the laziness technique to improve efficiency to $\tilde{\mathcal{O}}(n^2)$, by replacing the subroutine by **LazySample**$\mathbb{Z}_{m',m}$. We sketch a proof.

The floating-point computation $y_j = \sum_{i=1}^{n} x_i L_{j,i}$ with m bits of precision produces an error less than $\tilde{\mathcal{O}}(n^2 \left\|x\right\|_\infty \left\|L\right\|_\infty \epsilon)$ where $\epsilon = 2^{1-m}$. For $\tau = \tilde{\mathcal{O}}(\sqrt{n})$ we have that $\left\|x\right\|_\infty \leq \tau$ with overwhelming probability, and $\left\|L\right\|_\infty \leq \left\|L\right\|_S \leq s$ since $L^t L = C' \leq \sigma^2 \mathrm{Id}$. The error propagation is thus polynomial in n, and Lemma 1 ensures correction with the following parameters:

$$\tau = \mathcal{O}(\sqrt{n}), m = \mathcal{O}(n), m' = \mathcal{O}(\log n), \delta_p = \mathcal{O}(1/n^{5/2}).$$

Similarly to Lemma 2, one easily proves that, on average, less than $\mathcal{O}(n^2 \tau \delta_p)$ high-precision operations are performed, which in our case is $\mathcal{O}(1)$. Without fast integer arithmetic, the total complexity is thus less than $\mathcal{O}(n^2)\mathcal{O}(m'^2) + \mathcal{O}(1)\mathcal{O}(m^2) \leq \tilde{\mathcal{O}}(n^2)$.

6 Quasi-Linear Complexity in Ring Settings $\mathcal{R} = \mathbb{Z}_q[X]/(X^b \pm 1)$

For efficiency purposes, lattice cryptography often uses a special class of "algebraic" lattices arising from polynomial rings *i.e.* $\mathcal{R} = \mathbb{Z}[X]/(P(X))$ for some polynomial P of degree b. More precisely, the lattices are generated by an $\mathcal{R}$-basis, and can also be viewed as an integer lattice of dimension ℓb for some $\ell \geq 1$.

In this section, we show that for the ring settings $\mathcal{R} = \mathbb{Z}_q[X]/(X^b \pm 1)$, it is possible to achieve quasi-linear complexity using two improvement on top of our lazy variant of Peikert's offline phase [26,20]. The first improvement is to use special square-root algorithms (*e.g.* Babylonian Method or the Denman-Beavers iteration [8]) to preserve matrix structures, unlike Cholesky decomposition. In our case, we use block-circulant or block-skew-circulant structures, which are stable under transposition and multiplication, which implies that $\Sigma' = \Sigma - \eta^2 I_n = (s - \eta^2) I_n - B^t B$ has the same structure. The second improvement targets **Sample**$\mathbb{Z}$.

6.1 Structured Square-Root for $\mathcal{R} = \mathbb{Z}_q[X]/(X^b \pm 1)$

Consider the special ring setting $\mathcal{R} = \mathbb{Z}_q[X]/(X^b \pm 1)$, which includes $\mathbb{Z}_q[X]/(X^b - 1)$ for the class of NTRU lattices [13], and some cyclotomic lattices $\mathbb{Z}_q[X]/(\Phi_m)$ the m-th cyclotomic ring, when m is a power of two, made popular by the hardness results of [19].

When $P(X) = X^b - 1$ (resp. $P(X) = X^b + 1$) the integer representation $B \in \mathcal{M}^{bk \times bl}(\mathbb{Z})$ of any $\mathcal{R}$-basis is a b-block circulant, (resp. b-block skew-circulant) matrix, *i.e.* a matrix composed with $(b \times b)$-blocks of the form :

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_b \\ a_b & a_1 & \cdots & a_{b-1} \\ \vdots & \ddots & \ddots & \vdots \\ a_2 & \cdots & a_b & a_1 \end{bmatrix}, \quad \text{resp.} \quad \begin{bmatrix} a_1 & a_2 & \cdots & a_b \\ -a_b & a_1 & \cdots & a_{b-1} \\ \vdots & \ddots & \ddots & \vdots \\ -a_2 & \cdots & -a_b & a_1 \end{bmatrix}.$$

We denote these families by $\mathcal{C}_b$ (resp. $\mathcal{C}_b^\top$). These families are stable under ring operations (addition, product and inverse, when defined) because of the ring isomorphism with matrices over $\mathcal{R}$. Such isomorphisms also exist for other polynomials P, defining other b-block structures. However, circulant and skew-circulant structures have a key property for our improvement:

Fact 7 *Matrix families $\mathcal{C}_b$ and $\mathcal{C}_b^\top$ are stable under transposition.*

From this, we deduce that $\Sigma' = \Sigma - \eta^2 I_n = (s - \eta^2) I_n - B^t B \in \mathcal{C}_b$ (or $\mathcal{C}_b^\top$) when working in this ring setting. At this point, one would want to find a square root of Σ that is still structured. Interestingly, the solution can be found in algorithms that were designed to extract another notion of square root; namely, the Babylonian Method, or the Denman-Beavers iteration [8]. Indeed, those algorithms are searching for an Y such that $Y \cdot Y = X$, without symmetry requirement on X, and no guarentee of convergence in general. Lemma 3 proves that given as input $X \in \mathcal{S}_n^+$, such methods (quickly) converge to some $Y \in \mathcal{S}_n^+$ such that $Y^t \cdot Y = X$.

Definition. The Babylonian Method approximates the limit of the sequence:

$$Y_0(X) = I_n; \quad Y_{k+1}(X) = (Y_k(X) + X \cdot Y_k(X)^{-1})/2 \tag{2}$$

and if this sequence converges to an invertible limit $Y(X)$, it must verify $Y(X) = \frac{1}{2}(Y(X) + X \cdot Y(X)^{-1})$, which is equivalent to $Y(X) \cdot Y(X) = X$. The Denman-Beavers iteration is similar, using the sequences:

$$\begin{cases} Y_0(X) = X \\ Z_0(X) = \mathrm{Id} \end{cases} \qquad \begin{cases} Y_{k+1}(X) = \left(Y_k(X) + Z_k(X)^{-1}\right)/2 \\ Z_{k+1}(X) = \left(Z_k(X) + Y_k(X)^{-1}\right)/2 \end{cases} \tag{3}$$

it verifies the invariant $Y_k \cdot Z_k^{-1} = Z_k^{-1} \cdot Y_k = X$, and if it converges, the limit Y of Y_k verifies $Y \cdot Y = X$.

Lemma 3. *Let $X \in \mathcal{S}_n^+$ be a symmetric positive definite matrix, then the Babylonian Method, as defined by the sequence $Y_k(X)$ in (2) converges quadratically*[3] *to some $Y(X) \in \mathcal{S}_n^+$. Furthermore, if $X \in \mathcal{C}_b$ (resp. $\mathcal{C}_b^\top$) then $Y(X) \in \mathcal{C}_b$ (resp. $\mathcal{C}_b^\top$), which implies that $Y(X)^t Y(X) = X$. Similar results also hold for the Denman-Beavers iteration (3).*

Proof (sketch). By induction, write $Y_i(X)$ as QD_iQ^t for a fixed orthogonal matrix Q and diagonal matrices D_i. Each diagonal entry of (D_i) follows the Babylonian Square-Root sequence over $\mathbb{R}$, which allows to prove convergence. Structure preservation follows from ring and topological closure of $\mathcal{C}_b$ and $\mathcal{C}_b^\top$.

6.2 Improved Efficiency

Assuming the square root L of Σ was precomputed using one of the structure-preserving algorithms described below, each computation of $\mathbf{y} = \mathbf{x} \cdot L$ at precision m' can now be done in time $\tilde{\mathcal{O}}(nm'^2)$, but some coordinate may need to be recomputed at precision m. Using a similar analysis as in Sect. 5.2 with:

$$\tau = \mathcal{O}(\sqrt{n}), m = \mathcal{O}(n), m' = \mathcal{O}(\log n), \delta_p = \mathcal{O}(1/n^{7/2}).$$

we show that the "average" time[4] spent on the computation of $\mathbf{y} = x \cdot L$ is indeed $\tilde{\mathcal{O}}(n)$.

By combining Laziness and Structured-Square-Root, we move the complexity bottleneck to the **LazySample**$\mathbb{Z}$ subroutine, which is called n times and requires $\tilde{\mathcal{O}}(\tau) = \tilde{\mathcal{O}}(\sqrt{\lambda})$ trials in average. For $\lambda \sim n$, this leads to an overall average complexity of $\tilde{\mathcal{O}}(n^{1.5})$.

To reach quasi-linear complexity we need a third trick, detailed in the full version [9]. There, we improve the rejection sampling algorithm **Sample**$\mathbb{Z}$ so that it only needs a constant number of trials on average. This is done by sampling from a distribution before rejection which is much closer to the target distribution than the uniform distribution used in **Sample**$\mathbb{Z}$.

By combining the three techniques, we eventually obtain an implementation of Peikert's offline phase which runs in average[4] quasi-linear time. These results also apply to the recent variant of Micciancio and Peikert [20].

---

[3] The number of correct bits grows quadratically with the number k of iterations: $|s_k - s_\infty| \leq c\, 2^{-c'k^2}$ for some $c, c' > 0$

[4] We explain what we mean by average. As high-precision is triggered independently with small probability over n trials, the running times of the optimized Klein's Sampler and optimized Peikert's Offline Phase are bounded by some function $\tilde{\mathcal{O}}(n^2)$, except with negligible probability. However, when applying laziness in the ring setting, triggering high-precision once in the whole algorithm raises this instance's running time to $\tilde{\mathcal{O}}(n\lambda^2)$: only the average cost is below that bound. And dealing with average running times is less problematic in an offline phase, than in an online phase which is more subject to timing attacks.

Acknowledgements. We thank T. Lepoint, C. Peikert, O. Regev and D. Stehlé for useful discussions. We also thanks anonymous reviewers for their comments. Part of this work is supported by the Commission of the European Communities through the ICT program under contract ICT-2007-216676 ECRYPT II, and by China's 973 Program (Grant 2013CB834205).

References

1. S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In *Proc. EUROCRYPT '10*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572. Springer, 2010.
2. S. Agrawal, D. Boneh, and X. Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *Proc. CRYPTO '10*, volume 6223 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 2010.
3. M. Ajtai. Generating hard instances of lattice problems. In *Proc. STOC '96*, pages 99–108. ACM, 1996.
4. M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proc. ACM STOC '97*, pages 284–293, 1997.
5. L. Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.
6. X. Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In *Proc. PKC '10*, volume 6056 of *Lecture Notes in Computer Science*, pages 499–517. Springer, 2010.
7. D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. In *Proc. EUROCRYPT '10*, volume 6110 of *Lecture Notes in Computer Science*, pages 523–552. Springer, 2010.
8. E. Denman and A. Beavers. *The matrix sign function and computations in systems.* American Elsevier, 1976.
9. L. Ducas and P. Q. Nguyen. Faster Gaussian lattice sampling using lazy floating-point arithmetic. Full version of the ASIACRYPT '12 article.
10. L. Ducas and P. Q. Nguyen. Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures. In *Proc. ASIACRYPT '12*, Lecture Notes in Computer Science. Springer, 2012.
11. C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proc. STOC '09*, pages 169–178. ACM, 2009.
12. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proc. STOC '08*, pages 197–206. ACM, 2008.
13. J. Hoffstein, N. A. H. Graham, J. Pipher, J. H. Silverman, and W. Whyte. NTRUSIGN: Digital signatures using the NTRU lattice. In *Proc. of CT-RSA*, volume 2612 of *LNCS*. Springer-Verlag, 2003.
14. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory – Proc. ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.
15. P. N. Klein. Finding the closest lattice vector when it's unusually close. In *Proc. ACM SODA*, pages 937–941, 2000.
16. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Ann.*, 261:513–534, 1982.
17. R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA'11*, pages 319–339, 2011.

18. V. Lyubashevsky. Lattice signatures without trapdoors. *IACR Cryptology ePrint Archive*, 2011:537, 2011. In EUROCRYPT '12.
19. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230, 2010. In EUROCRYPT '10.
20. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. *IACR Cryptology ePrint Archive*, 2011:501, 2011. In EUROCRYPT '12.
21. D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:372–381, 2004.
22. D. Micciancio and O. Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, Berlin, 2009.
23. I. Morel, D. Stehlé, and G. Villard. H-LLL: using householder inside LLL. In *Proc. ISSAC '09*, pages 271–278. ACM, 2009.
24. P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. *J. Cryptology*, 22(2):139–160, 2009.
25. P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. Comput.*, 39(3):874–903, 2009.
26. C. Peikert. An efficient and parallel Gaussian sampler for lattices. In *Proc. CRYPTO '10*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2010.
27. O. Regev. The learning with errors problem (invited survey). In *Proc. IEEE Conference on Computational Complexity*, pages 191–204. IEEE Computer Society, 2010.
28. M. Rückert and M. Schneider. Estimating the security of lattice-based cryptosystems. Cryptology ePrint Archive, Report 2010/137, 2010. `http://eprint.iacr.org/`.
29. C.-P. Schnorr. A more efficient algorithm for lattice basis reduction. *J. Algorithms*, 9(1):47–62, 1988.