Quadratic Time, Linear Space Algorithms for Gram-Schmidt Orthogonalization and Gaussian Sampling in Structured Lattices

Vadim Lyubashevsky, Thomas Prest

Quadratic Time, Linear Space Algorithms for Gram-Schmidt Orthogonalization and Gaussian Sampling in Structured Lattices

Vadim Lyubashevsky[1,2⋆] and Thomas Prest[2,3]

[1] INRIA
[2] École Normale Supérieure
{vadim.lyubashevsky,thomas.prest}@ens.fr
[3] Thales Communications & Security

Abstract. A procedure for sampling lattice vectors is at the heart of many lattice constructions, and the algorithm of Klein (SODA 2000) and Gentry, Peikert, Vaikuntanathan (STOC 2008) is currently the one that produces the shortest vectors. But due to the fact that its most time-efficient (quadratic-time) variant requires the storage of the Gram-Schmidt basis, the asymptotic space requirements of this algorithm are the same for general and ideal lattices. The main result of the current work is a series of algorithms that ultimately lead to a sampling procedure producing the same outputs as the Klein/GPV one, but requiring only linear-storage when working on lattices used in ideal-lattice cryptography. The reduced storage directly leads to a reduction in key-sizes by a factor of $\Omega(d)$, and makes cryptographic constructions requiring lattice sampling much more suitable for practical applications.

At the core of our improvements is a new, faster algorithm for computing the Gram-Schmidt orthogonalization of a set of vectors that are related via a *linear isometry*. In particular, for a linear isometry $r : \mathbb{R}^d \to \mathbb{R}^d$ which is computable in time $O(d)$ and a d-dimensional vector $\mathbf{b}$, our algorithm for computing the orthogonalization of $(\mathbf{b}, r(\mathbf{b}), r^2(\mathbf{b}), \ldots, r^{d-1}(\mathbf{b}))$ uses $O(d^2)$ floating point operations. This is in contrast to $O(d^3)$ such operations that are required by the standard Gram-Schmidt algorithm. This improvement is directly applicable to bases that appear in ideal-lattice cryptography because those bases exhibit such "isometric structure". The above-mentioned algorithm improves on a previous one of Gama, Howgrave-Graham, Nguyen (EUROCRYPT 2006) which used different techniques to achieve only a constant-factor speed-up for similar lattice bases. Interestingly, our present ideas can be combined with those from Gama et al. to achieve an even an larger practical speed-up.

We next show how this new Gram-Schmidt algorithm can be applied towards lattice sampling in quadratic time using only linear space. The main idea is that rather than pre-computing and storing the Gram-Schmidt vectors, one can compute them "on-the-fly" while running the

⋆ This research was partially supported by the ANR JCJC grant "CLE".

sampling algorithm. We also rigorously analyze the required arithmetic precision necessary for achieving negligible statistical distance between the outputs of our sampling algorithm and the desired Gaussian distribution. The results of our experiments involving NTRU lattices show that the practical performance improvements of our algorithms are as predicted in theory.

1 Introduction

Sampling lattice points is one of the fundamental procedures in lattice cryptography. It is used in hash-and-sign signatures [GPV08], (hierarchical) identity-based encryption schemes [GPV08,CHKP10,ABB10], standard-model signatures [ABB10,Boy10], attribute-based encryption [BGG$^+$14], and many other constructions. Being able to output shorter vectors leads to more secure schemes, and the algorithm that produces the currently-shortest samples is the randomized version of Babai's nearest-plane algorithm [Bab86] due to Klein [Kle00] and Gentry, Peikert, Vaikuntanathan [GPV08].

The main inefficiency of cryptography based on general lattices is that the key size is usually (at least) quadratic in the security parameter, which is related to the fact that a d-dimensional lattice is generated by d vectors. For security, the lattice dimension is usually taken to be on the order of 512, and this results in keys that are larger than one megabyte in size and unsuitable for most real-world applications. For this reason, all practical implementations of lattice schemes (e.g. [HPS98,LMPR08,LPR13a,DDLL13,DLP14]) rely on the hardness of problems involving polynomial rings [PR06,LM06,LPR13a], in which lattices can be represented by a few polynomials. Due to the fact that solving certain average-case problems over polynomial rings was shown to be as hard as solving worst-case problems in *ideal lattices* [SSTX09,LPR13a], building cryptographic systems using polynomimals is often referred to as ideal lattice cryptography.

During its execution, the Klein/GPV algorithm is implicitly computing the Gram-Schmidt orthogonalization of the input basis.[1] Since the Gram-Schmidt procedure requires $\Theta(d^3)$ operations, the Klein/GPV sampler also requires at least this much time. For improving the time-complexity, one can pre-compute and store the Gram-Schmidt basis, which results in a sampling procedure that uses only $\Theta(d^2)$ operations. The Gram-Schmidt basis, however, requires the storage of $\Theta(d^2)$ elements, and so the key size is, as mentioned above, unacceptably large. One may hope that, again, using polynomials results in a decrease of the required storage. In this case, unfortunately, such rings do not help. The Gram-Schmidt orthogonalization procedure completely destroys the nice structure of polynomial lattices. So while a polynomial lattice basis can be represented by a few vectors, its Gram-Schmidt basis will have d vectors. Thus the $\Theta(d^2)$-operation Klein/GPV algorithm requires as much storage when using polyno-

[1] The Gram-Schmidt procedure produces a mutually-orthogonal set of vectors $(\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_d)$ that span the same inner-product space as the input vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$

mial lattices as when using general lattices, and is equally unsuitable for practical purposes. Therefore the only realistic solution is to not store the pre-processed Gram-Schmidt basis, which would then allow for the polynomial lattice algorithm to be linear-space (since only the original, compact-representation basis needs to be stored), but require at least $\Omega(d^3)$ time due to the fact that the Gram-Schmidt basis will need to be computed.

1.1 Our Results

Our main result is an algorithm that computes the Gram-Schmidt basis of certain algebraic lattices using $\Theta(d^2)$, instead of $\Theta(d^3)$, arithmetic operations. We then show how this new procedure can be combined with the Klein/GPV sampler to achieve a "best-of-both-worlds" result – a sampling algorithm that requires $\Theta(d^2)$ operations, which does not require storing a pre-processed Gram-Schmidt basis. In ideal lattice cryptography, this implies being able to have keys that consist of just the compact algebraic basis requiring only linear storage. Not pre-computing the Gram-Schmidt basis of course necessarily slows down our sampling algorithm versus the one where this basis is already stored in memory. But our new orthogonalization algorithm is rather efficient, and the slowdown in some practical applications is by less than a factor of 4 (see the performance table in Section 9.1). In case of amortization (i.e. sampling more than one vector at a time), the running time of our new algorithm becomes essentially the same as that of the one requiring the storage of the orthogonalized basis. As a side note, since the Klein/GPV algorithm is just a randomized version of the classic Babai nearest plane algorithm [Bab86], all our improvements apply to the latter as well.

While analyzing the running-time of lattice algorithms, it is very important to not only consider the number of arithmetic operations, but also the arithmetic precision required for the algorithms to be stable. The run-time of algorithms that are not numerically stable may suffer due to the high precision required during their execution. A second important issue is understanding how the precision affects the statistical closeness of the distribution of the outputted vectors versus the desired distribution. We rigorously show that in order to have statistical closeness between the output distribution and the desired discrete Gaussian one be at most $2^{-\lambda}$, the required precision of the Gram-Schmidt basis needs to be a little more (in practice, less than a hundred bits) than λ. We then experimentally show that our new Gram-Schmidt procedure is rather numerically stable, and the intermediate storage is not much more than the final required precision.[2]

[2] The reason that we only do experimental analysis of this second part, rather than a rigorous theoretical one, is because the precision required in practice will be much less than in theory due to the fact that theoretically, there could exist bases on which the Gram-Schmidt procedure is rather unstable. When using this procedure in cryptographic schemes, we can always test our basis for numerical stability by comparing the output of the exact Gram-Schmidt algorithm versus the "approximate" Gram-Schmidt algorithm that will be actually used.

A third issue that also needs to be considered in practice is the space requirements of the algorithm during run-time. While the stored basis is very short, it could be that the intermediate computations require much larger storage (e.g. if the intermediate computation requires storing the entire Gram-Schmidt basis to a high precision). Our sampling algorithm, however, is rather efficient in this regard because it only requires storing one Gram-Schmidt *vector* at a time. The storage requirement during run-time is therefore less than 64KB for typical dimensions used in cryptographic applications.

Isometries and Ideal Lattices. Interestingly, our improved orthogonalization algorithm for polynomial lattices does not have much to do with their algebraic structure, but rather relies on their implicit geometric properties. The types of bases whose Gram-Schmidt orthogonalization we speed up are those that consist of a set of vectors that are related via a *linear isometry*. In particular, if H is a d-dimensional Hermitian inner-product space and $r : H \to H$ is a linear map that preserves the norm and is computable using $\mathcal{O}(d)$ operations, then we show (both theoretically and via implementations) that orthogonalizing a set of vectors $\{\mathbf{b}, r(\mathbf{b}), r^2(\mathbf{b}) \ldots, r^{d-1}(\mathbf{b})\}$ can be done using $\Theta(d^2)$ floating point operations.

We now explain the connection between isometries and ideal lattices. Consider the cyclotomic number field, with the usual polynomial addition and multiplication operations, $F = \mathbb{Q}[X]/\langle \Phi_m(X) \rangle$ where $\Phi_m(X)$ is the m^{th} cyclotomic polynomial (and so it has degree $\phi(m)$). Elements in F can be represented via a *canonical embedding*[3] into $\mathbb{C}^{\phi(m)}$, and in that case F becomes isomorphic, as an inner product space, to $\mathbb{R}^{\phi(m)}$ where the inner product $\langle \mathbf{a}, \mathbf{b} \rangle$ of $\mathbf{a}, \mathbf{b} \in F$ is defined as

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{1 \le i \le m, gcd(i,m)=1} \overline{\mathbf{a}(\zeta_m^i)} \cdot \mathbf{b}(\zeta_m^i),$$

where $\zeta_m \in \mathbb{C}$ is an m^{th} root of unity (c.f. [LPR13b, Sections 2.2, 2.5.2]).[4]

With the above definition of inner product (which is in fact the usual inner product over $\mathbb{C}^{\phi(m)}$ when elements in F are represented via the canonical embedding), the norm of an element $\mathbf{b} \in F$ is

$$\|\mathbf{b}\| = \sqrt{\sum_{1 \le i \le m, gcd(i,m)=1} |\mathbf{b}(\zeta_m^i)|^2}.$$

[3] The canonical embedding of a polynomial $\mathbf{b} \in F$ is a vector in $\mathbb{C}^{\phi(m)}$ whose coefficients are the evaluations of $\mathbf{b}$ on each of the $\phi(m)$ complex roots of $\Phi_m(X)$. Due to the fact that half of the roots of $\Phi_m(X)$ are conjugates of the other half, the resulting embedded vector in $\mathbb{C}^{\phi(m)}$ can be represented by $\phi(m)$ real numbers.

[4] We point out that the actual computation of the inner product does not require any operations over $\mathbb{C}$. The reason is that $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{1 \le i \le m, gcd(i,m)=1} \overline{\mathbf{a}(\zeta_m^i)} \cdot \mathbf{b}(\zeta_m^i)$ can be rewritten as $\overline{(V\mathbf{a})^T} V\mathbf{b} = \mathbf{a}^T \overline{V^T} V\mathbf{b}$ for a Vandermonde matrix V with coefficients in $\mathbb{C}$. The matrix $\overline{V^T}V$, however, is a simple *integer* matrix, multiplication by which can be performed in linear time for most "interesting" cyclotomic polynomials (e.g. m is prime or a power of 2).

Since all ζ_m^i, where $gcd(i, m) = 1$, are roots of $\Phi_m(X)$ and $|\zeta_m^i| = 1$, one can check that for any $\mathbf{b} \in F$, $\|\mathbf{b}\| = \|\mathbf{b}X\|$. Since the function $r : F \to F$ defined as $r(\mathbf{b}) = \mathbf{b}X$ is linear, it is also an isometry (since it preserves the norm). Furthermore, since F is a field, for any non-zero $\mathbf{b} \in F$, the elements $\mathbf{b}, \mathbf{b}X, \mathbf{b}X^2, \ldots, \mathbf{b}X^{\phi(m)-1}$ are all linearly-independent. When $\mathbf{b}$ is an element of $R = \mathbb{Z}[X]/\langle \Phi_m(X) \rangle$, the set

$$\{\mathbf{b}, \mathbf{b}X, \mathbf{b}X^2, \ldots, \mathbf{b}X^{\phi(m)-1}\} = \{\mathbf{b}, r(\mathbf{b}), r^2(\mathbf{b}), \ldots, r^{\phi(m)-1}(\mathbf{b})\}$$

therefore generates the ideal $\langle \mathbf{b} \rangle$ as an additive group. Such bases containing short elements can serve as private keys in cryptographic schemes.[5]

Paper Organization. In Section 2, we set up the notations and definitions that will be used throughout the paper. In Section 3.1, we describe a simple version of our new algorithm that efficiently orthogonalizes a given set of vectors, and in Section 3.2 we give the full, optimized algorithm. In Section 4, we describe an algorithm that, given the orthogonalization, returns the transformation matrix μ that converts the set $\{\mathbf{b}, r(\mathbf{b}), \ldots, r(\mathbf{b})\}$ to $\{\tilde{\mathbf{b}}_1, \tilde{\mathbf{b}}_2, \ldots, \tilde{\mathbf{b}}_n\}$. In Section 5, we extend our basic algorithms to those that can more efficiently orthogonalize sets of vectors of the form $\mathbf{b}_1, r(\mathbf{b}_1), \ldots, r^{n-1}(\mathbf{b}_1), \mathbf{b}_2, r(\mathbf{b}_2), \ldots, r^{n-1}(\mathbf{b}_2), \mathbf{b}_3, r(\mathbf{b}_3), \ldots$. These types of sets are the ones that normally occur as secret keys in lattice cryptography. A particular example of such a set is the NTRU lattice, which we discuss in Section 7. In that section, we also give timing comparisons between the *exact* version of our orthogonalization algorithm (which is analyzed in Section 6), and that of [GHN06], for computing the Gram-Schmidt orthogonalization of NTRU lattices. Since the two algorithms use different techniques to achieve speed-ups, we demonstrate that the two improvements can complement each other in the form of an even faster algorithm. In Section 8, we show how to implement Babai's nearest plane algorithm and the Klein/GPV sampling in linear space for lattices whose basis contains vectors that are related via an isometry. In Section 9 we focus on the implementation aspects of our results. In particular, we analyze the required precision to insure the correct functionality of our sampling algorithm.

1.2 Related Work

Computing faster orthogonalization for vectors that are somehow related has been considered in the past. For example, Sweet [Swe84] demonstrated an algorithm that orthogonalizes $d \times d$ Toeplitz matrices using $O(d^2)$ operations. This is the same linear-time speed-up as for our algorithm, but for a different class of

[5] Normally, the bases used in schemes have slightly different forms, such as consisting of a concatenation of elements from R, or being formed by several elements in R. Such bases still contain large components that are related via an isometry, and we discuss this in more detail in Section 7.

structured matrices.[6] The techniques in that paper seem to be rather different than in ours – [Swe84] works with the concrete representation of Toeplitz matrices, whereas we only rely on the abstract geometric properties of isometries.

For the special case of NTRU lattices, Gama, Howgrave-Graham, and Nguyen [GHN06] devised algorithms that take advantage of a structure of NTRU bases called *symplecticity*. This allows them to be faster (by a constant factor) than standard Gram-Schmidt orthogonalization when performing orthogonalization in exact arithmetic. We adapt our algorithms for the same application and they outperform those from [GHN06].[7] And since our algorithm and that of [GHN06] relies on different ideas, it turns out that we can combine the two techniques to achieve a greater overall improvement (see Figure 1 in Section 7).

2 Preliminaries

2.1 Notations

Throughout the paper, we will be working over a d-dimensional inner product space H (usually $H = \mathbb{R}^d$ or $\mathbb{C}^d$), with $\langle \cdot, \cdot \rangle$ and $\|\cdot\|$ being a scalar product and the associated norm over H. Except when stated otherwise, vectors will be written in bold, matrices and bases in capital bold, and scalars in non-bold letters. $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$ will be either an ordered set of independent vectors, also called a basis, or the $n \times d$ matrix whose rows are the $\mathbf{b}_i$. We denote $\mathbf{B}_k = Span(\mathbf{b}_1, ..., \mathbf{b}_k)$ to be the vector space spanned by the vectors $\mathbf{b}_1, ..., \mathbf{b}_k$.

Definition 2.1. *A linear isometry is a linear map $r : H \to H$ such that for any $\mathbf{x}, \mathbf{y} \in H$:*

$$\langle r(\mathbf{x}), r(\mathbf{y}) \rangle = \langle \mathbf{x}, \mathbf{y} \rangle,$$

or equivalently

$$\|\mathbf{x}\| = \|r(\mathbf{x})\|.$$

For conciseness, we will sometimes say isometry instead of linear isometry. Since the dimension of H is finite, it is immediate that r is invertible. We will be assuming throughout the work that both r and r^{-1} are computable in time $\mathcal{O}(d)$.

Definition 2.2. *Let $\mathbf{x} \in H$, and F be a subspace of H. Then $\mathbf{Proj}(\mathbf{x}, F)$, the projection of $\mathbf{x}$ over F, is the unique vector $\mathbf{y} \in F$ such that $\|\mathbf{x}-\mathbf{y}\| = \min_{\mathbf{z} \in F} \|\mathbf{x}-\mathbf{z}\|$*

[6] One may imagine that it may be possible to somehow adapt the results of [Swe84] to the orthogonalization of bases of ideal lattices. The idea would be to embed elements of $F = \mathbb{Q}[X]/\langle \Phi_m(X) \rangle$ into $C = \mathbb{Q}[X]/\langle X^m - 1 \rangle$ and then try to use the fact that in the coefficient representation, the elements $\mathbf{b}, \mathbf{b}X, \mathbf{b}X^2, \ldots$ in C form a Toeplitz matrix. One would have to also take care to make sure that the norm in F corresponds to the coefficient norm in C. We are not sure whether this direction is viable, and even if it is, the algorithm would necessarily involve computations over dimension m, rather than $\phi(m)$, which would unnecessarily increase the running-time of all algorithms.

[7] We mention that [GHN06] also contains other results which are independent of Gram-Schmidt orthogonalization and are therefore not improved by our work.

Proposition 2.3. *Let r be an isometry and and F be a subspace of H. Then :*
1. $\mathbf{x} \perp F \Rightarrow r(\mathbf{x}) \perp r(F)$
2. $r(\mathbf{Proj}(\mathbf{x}, F)) = \mathbf{Proj}(r(\mathbf{x}), r(F))$

Proof. We prove the two claims separately :
1. Since r preserves the dot product, it also preserves orthogonality between vectors.
2. r preserves the norm, so

$$\|\mathbf{x} - \mathbf{y}\| = \min_{\mathbf{z} \in F} \|\mathbf{x} - \mathbf{z}\| \Longrightarrow \|r(\mathbf{x}) - r(\mathbf{y})\| = \min_{\mathbf{z} \in r(F)} \|r(\mathbf{x}) - \mathbf{z}\|$$

□

2.2 The Gram-Schmidt Orthogonalization

We provide three equivalent definitions of the Gram-Schmidt orthogonalization of an ordered basis. The first one is geometrical, the second one is a mathematical formula, and the third one looks at each vector of the basis as its decomposition over two orthogonal vector spaces. Although the two first definitions are standard and useful for comprehension and computation, the third one is less common and we will mostly use it to prove that a basis is indeed the orthogonalization of another one.

Definition 2.4. *Let the basis $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$ be an ordered set of vectors in H. Its Gram-Schmidt orthogonalization (GSO) is the unique basis $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$ verifying one of these properties :*

- $\forall k \in [\![1, n]\!], \tilde{\mathbf{b}}_k = \mathbf{b}_k - \mathbf{Proj}(\mathbf{b}_k, \mathbf{B}_{k-1})$
- $\forall k \in [\![1, n]\!], \tilde{\mathbf{b}}_k = \mathbf{b}_k - \sum_{j=1}^{k-1} \frac{\langle \mathbf{b}_k, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2} \tilde{\mathbf{b}}_j$
- $\forall k \in [\![1, n]\!], \tilde{\mathbf{b}}_k \perp \mathbf{B}_{k-1}$ *and* $(\mathbf{b}_k - \tilde{\mathbf{b}}_k) \in \mathbf{B}_{k-1}$

The bases $\mathbf{B}$ and $\tilde{\mathbf{B}}$ then satisfy : $\forall k \in [\![1, n]\!], \mathbf{B}_k = \tilde{\mathbf{B}}_k$. For a vector $\mathbf{b}_k$, we will say that $\tilde{\mathbf{b}}_k$ is its Gram-Schmidt reduction (GSR).

Algorithm 1 describes the Gram-Schmidt process as it is usually presented.

2.3 The Gram-Schmidt and LQ Decompositions

The Gram-Schmidt decomposition (GSD) is a natural side-product of the Gram-Schmidt orthogonalization which gives the relation between the input and output bases in the form of a matrix μ. Such a matrix is useful in many cases – for example, it is crucially used in the LLL algorithm [LLL82]. Outside cryptography, applications include solving undetermined linear systems, least square problems and computing eigenvalues.

Algorithm 1 `GramSchmidt_Process`$(\mathbf{B})$

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$
Ensure: Gram-Schmidt reduced basis $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$
1: **for** $i = 1, ..., n$ **do**
2: $\quad \tilde{\mathbf{b}}_i \leftarrow \mathbf{b}_i$
3: $\quad$ **for** $j = 1, ..., i-1$ **do**
4: $\quad\quad \mu_{i,j} = \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2}$
5: $\quad\quad \tilde{\mathbf{b}}_i \leftarrow \tilde{\mathbf{b}}_i - \mu_{i,j}\tilde{\mathbf{b}}_j$
6: $\quad$ **end for**
7: **end for**
8: **return** $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$

Definition 2.5 (Gram-Schmidt Decomposition). *Let* $\mathbf{B}$ *be a* $d \times n$ *matrix.* $\mathbf{B}$ *can be uniquely decomposed as* $\mathbf{B} = \mu \times \tilde{\mathbf{B}}$*, where* $\tilde{\mathbf{B}}$ *is the GSO of* $\mathbf{B}$ *and* $\mu = (\mu_{i,j})_{1 \leqslant i,j \leqslant n}$ *is the lower triangular matrix such that*

$$\mu_{i,j} = \begin{cases} \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2} & \text{if } i > j \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Notice that the matrix μ is automatically constructed in Algorithm 1 while computing the GSO. This, however, will not be the case in our improved GSO algorithm, and this is why in this paper we will differentiate between GSO and GSD.

We now recall the definition of LQ decomposition and give its natural relation to the $\mathbf{B} = \mu \times \tilde{\mathbf{B}}$ decomposition.

Definition 2.6 (LQ decomposition). *Let* $\mathbf{B}$ *be a square invertible matrix.* $\mathbf{B}$ *can be decomposed as* $\mathbf{B} = L \times Q$*, where* L *is a lower triangular matrix and* Q *is an orthonormal matrix. If we request the diagonal coefficients of* L *to be positive, then this decomposition is unique.*

Fact 2.7 *Let* $\mathbf{B}$ *be a square invertible matrix, and* $\mathbf{B} = \mu \times \tilde{\mathbf{B}}$ *its GSD. An LQ decomposition of* $\mathbf{B}$ *can be computed in time* $\mathcal{O}(n^2)$ *by taking* $Q = D^{-1} \times \tilde{\mathbf{B}}$ *and* $L = \mu \times D$*, where* $D = Diag(\|\tilde{\mathbf{b}}_1\|, ..., \|\tilde{\mathbf{b}}_n\|)$*.*

2.4 Discrete Gaussians

Discrete Gaussians are n-dimensional Gaussians discretized over some lattice Λ.

Definition 2.8. *The* n*-dimensional Gaussian function* $\rho_{\sigma,\boldsymbol{c}} : \mathbb{R}^n \to (0, 1]$ *of standard deviation* σ *and center* $\mathbf{c}$ *is defined by :*

$$\rho_{\sigma,\mathbf{c}}(\boldsymbol{x}) \stackrel{\Delta}{=} \frac{1}{(\sigma\sqrt{2\pi})^n} \exp\left(-\frac{\|\boldsymbol{x} - \mathbf{c}\|^2}{2\sigma^2}\right)$$

For any lattice $\Lambda \subset \mathbb{R}^n$, $\rho_{\sigma,\mathbf{c}}(\Lambda) \stackrel{\Delta}{=} \sum_{\boldsymbol{x}\in\Lambda} \rho_{\sigma,\mathbf{c}}(\boldsymbol{x})$. Normalizing $\rho_{\sigma,\mathbf{c}}(\boldsymbol{x})$ by $\rho_{\sigma,\mathbf{c}}(\Lambda)$, we obtain the probability distribution function of the discrete Gaussian distribution $D_{\Lambda,\sigma,\mathbf{c}}$.

Gaussian Sampling was introduced in [Kle00,GPV08] as a technique to sample from discrete Gaussian distributions, and has since found numerous applications in lattice-based cryptography. In [GPV08], it requires a basis $\mathbf{B}$ of the lattice Λ being sampled from, as well as the GSO $\tilde{\mathbf{B}}$ of $\mathbf{B}$.

3 Gram-Schmidt Orthogonalization over Isometric Bases

In this section we present our improved isometric Gram-Schmidt algorithm. In Section 3.1, we present a first simple version which we believe is very intuitive to understand, and then present a slightly faster, more involved, version of it in Section 3.2.

Definition 3.1. *Let $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$ be an ordered basis of a lattice $\Lambda \subseteq H$. We say that $\mathbf{B}$ is isometric if there exists an isometry r such that*

$$\forall k \in [\![2, n]\!], \mathbf{b}_k = r(\mathbf{b}_{k-1})$$

3.1 A Quadratic-Time algorithm

We now describe a simple algorithm that computes the GSO of any isometric basis in time $\Theta(nd)$ (or $\Theta(n^2)$ when $n = d$).

We briefly expose the general idea behind the algorithm before presenting it formally. If $\tilde{\mathbf{b}}_k$ is the GSR of $\mathbf{b}_k$, then $r(\tilde{\mathbf{b}}_k)$ is almost the GSR of $\mathbf{b}_{k+1}$: it is orthogonal to $\mathbf{b}_2, ..., \mathbf{b}_k$, but not to $\mathbf{b}_1$. However, reducing $r(\tilde{\mathbf{b}}_k)$ with respect to $\mathbf{b}_1$ would break its orthogonality to $\mathbf{b}_2, ..., \mathbf{b}_k$, so what we really need to do is to reduce it with respect to $\mathbf{b}_1 - \mathbf{Proj}(\mathbf{b}_1, Span(\mathbf{b}_2...\mathbf{b}_k))$. Indeed, this latter vector is orthogonal to $\mathbf{b}_2, ..., \mathbf{b}_k$, so reducing $r(\tilde{\mathbf{b}}_k)$ with respect to it won't break the orthogonality of $r(\tilde{\mathbf{b}}_k)$ to $\mathbf{b}_2, ..., \mathbf{b}_k$. Fortunately, $\mathbf{b}_1 - \mathbf{Proj}(\mathbf{b}_1, Span(\mathbf{b}_2...\mathbf{b}_k))$ can itself be updated quickly. Definition 3.2 and Algorithm 2 formalize this idea.

Definition 3.2. *Let $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$ be an ordered basis and $k \in [\![1, n]\!]$. We denote $\mathbf{v}_{\mathbf{B},k} = \mathbf{b}_1 - \mathbf{Proj}(\mathbf{b}_1, r(\mathbf{B}_{k-1}))$. When $\mathbf{B}$ is obvious from the context, we simply write $\mathbf{v}_k$.*

Proposition 3.3. *Let $\mathbf{B}$ be an isometric basis with respect to r. Algorithm 2 returns the GSO of $\mathbf{B}$. Moreover, if $r(\mathbf{v})$ can be computed in time $O(d)$ for any $\mathbf{v} \in H$, then Algorithm 2 terminates in time $O(nd)$.*

Proof. We first prove the correctness of the scheme by proving by induction that for every $k \in [\![1, n]\!]$, we have the following :

Algorithm 2 `Isometric_GSO`($\mathbf{B}$)

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$
Ensure: Gram-Schmidt reduced basis $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$
1: $\tilde{\mathbf{b}}_1 \leftarrow \mathbf{b}_1$
2: $\mathbf{v}_1 \leftarrow \mathbf{b}_1$
3: **for** $k = 1, ..., n-1$ **do**
4: $\quad \tilde{\mathbf{b}}_{k+1} \leftarrow r(\tilde{\mathbf{b}}_k) - \frac{\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle}{\|\mathbf{v}_k\|^2}\mathbf{v}_k$
5: $\quad \mathbf{v}_{k+1} \leftarrow \mathbf{v}_k - \frac{\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle}{\|\tilde{\mathbf{b}}_k\|^2} r(\tilde{\mathbf{b}}_k)$
6: **end for**
7: **return** $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$

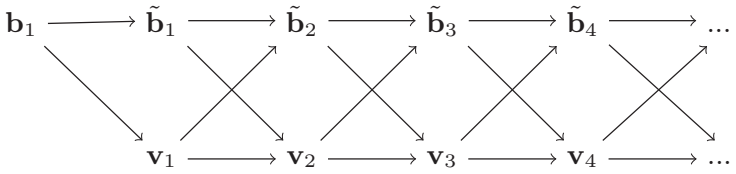

Fig. 1. Computing all the orthogonalized vectors from the first one in Algorithm 2

- $\mathbf{v}_k = \mathbf{b}_1 - \mathbf{Proj}(\mathbf{b}_1, r(\mathbf{B}_{k-1}))$ (1)
- $\tilde{\mathbf{b}}_k = \mathbf{b}_k - \mathbf{Proj}(\mathbf{b}_k, \mathbf{B}_{k-1})$ (2)

This is trivially true for $k = 1$. Assuming (1) and (2) are true at step k, we have:

- Since $\mathbf{v}_k$ and $\tilde{\mathbf{b}}_k$ are already orthogonal to $r(\mathbf{B}_{k-1})$, $\mathbf{v}_{k+1}$ also is as a linear combination of the two. But $\mathbf{v}_{k+1}$ is also the orthogonalization of $\mathbf{v}_k$ w.r.t. $r(\tilde{\mathbf{b}}_k)$, so it is orthogonal to $r(\mathbf{B}_{k-1}) + Span(r(\tilde{\mathbf{b}}_k)) = r(\mathbf{B}_k)$. On the other hand, $\mathbf{b}_1 - \mathbf{v}_k$ is in $r(\mathbf{B}_{k-1})$ so $\mathbf{b}_1 - \mathbf{v}_{k+1}$ is in $r(\mathbf{B}_k)$. By applying Definition 2.4, we can conclude that (1) is true for $k+1$.
- The same reasoning holds for $\tilde{\mathbf{b}}_{k+1}$: it is orthogonal to $r(\mathbf{B}_{k-1})$ because both $\mathbf{v}_k$ and $r(\tilde{\mathbf{b}}_k)$ are. But since it also is orthogonalized w.r.t. $\mathbf{v}_k$ (in line 4 of the algorithm), it then is orthogonal to $r(\mathbf{B}_{k-1}) + Span(\mathbf{v}_k) = \mathbf{B}_k$. On the other hand, $\mathbf{b}_{k+1} - \tilde{\mathbf{b}}_{k+1} = r(\mathbf{b}_k - \tilde{\mathbf{b}}_k) + \frac{\langle r(\tilde{\mathbf{b}}_k), \mathbf{v}_k\rangle}{\langle \mathbf{v}_k, \mathbf{v}_k\rangle}\mathbf{v}_k$ is in $\mathbf{B}_k$. As before, we can conclude that (2) is true for $k+1$.

Since (2) is verified for any $k \in [\![1, n]\!]$, $\tilde{\mathbf{B}}$ is the GSO of $\mathbf{B}$.

The time complexity of the algorithm is straightforward : assuming additions, subtractions, multiplications and divisions are done in constant time, each scalar product or square norm takes time $\mathcal{O}(d)$. Since there are $3(n-1)$ norms or scalar products, and $2(n-1)$ computations of $r(.)$, the total complexity is $\mathcal{O}(nd)$. □

3.2 Making Isometric GSO Faster

Algorithm 2 is already $O(n)$ times faster than the classical Gram-Schmidt process. In this subsection, we show that intermediate values are strongly interde-

pendent and that this fact can be used to speed up our GSO implementation by about 67%.

Lemma 3.4. *Let* $\mathbf{B}$ *be an isometric basis. For any* k *in* $[\![1, n]\!]$*, we have the following equalities:*
- $\langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_k)\rangle = \langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle$
- $\|\mathbf{v}_k\|^2 = \langle \mathbf{v}_k, \mathbf{v}_1\rangle = \|\tilde{\mathbf{b}}_k\|^2$

When implicit from context, we will denote $C_k = \langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle$ *and* $D_k = \|\tilde{\mathbf{b}}_k\|^2$*. We have the following recursive formula :*

$$\forall k \in [\![1, n-1]\!], D_{k+1} = D_k - \frac{C_k^2}{D_k}$$

Proof. We prove each of the three equalities separately :
- The equality $\langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_k)\rangle = \langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle$ is equivalent to $\langle \mathbf{v}_k - \mathbf{v}_1, r(\tilde{\mathbf{b}}_k)\rangle = 0$, which is true since $\mathbf{v}_k - \mathbf{v}_1 = \mathbf{Proj}(\mathbf{b}_1, r(\mathbf{B}_{k-1}))$ is in the subspace $r(\mathbf{B}_{k-1})$ and $\tilde{\mathbf{b}}_k$ is orthogonal to $r(\mathbf{B}_{k-1})$
- The equality $\|\mathbf{v}_k\|^2 = \langle \mathbf{v}_k, \mathbf{v}_1\rangle$ is obtained by following the same reasoning as above
- The equality $\|\mathbf{v}_k\|^2 = \|\tilde{\mathbf{b}}_k\|^2$ is shown by induction : it is the case for $k = 1$. By observing that $\tilde{\mathbf{b}}_{k+1}$ is orthogonal to $\mathbf{v}_k$ from line 4 of Algorithm 2(resp. $\mathbf{v}_{k+1}$ is orthogonal to $r(\mathbf{b}_k)$ from line 5)), we can use the Pythagorean theorem to compute $\|\tilde{\mathbf{b}}_{k+1}\|^2$ and $\|\mathbf{v}_{k+1}\|^2$:

$$\|\tilde{\mathbf{b}}_{k+1}\|^2 = \|\tilde{\mathbf{b}}_k\|^2 - \frac{\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle^2}{\|\mathbf{v}_k\|^2} \quad \text{and} \quad \|\mathbf{v}_{k+1}\|^2 = \|\mathbf{v}_k\|^2 - \frac{\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle^2}{\|\tilde{\mathbf{b}}_k\|^2}$$

At which point we can conclude by induction that $\|\mathbf{v}_{k+1}\|^2 = \|\tilde{\mathbf{b}}_{k+1}\|^2$, and these equalities also yield the recursive formula $D_{k+1} = D_k - \frac{C_k^2}{D_k}$.

□

This result allows us to speed up further the GSO for isometric bases. At each iteration of the algorithm `Isometric_GSO`, instead of computing $\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle$, $\|\tilde{\mathbf{b}}_k\|^2$ and $\|\mathbf{v}_k\|^2$, one only needs to compute $\langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_k)\rangle$, and can instantly compute $\|\tilde{\mathbf{b}}_k\|^2 = \|\mathbf{v}_k\|^2$ from previously known values. We choose $\langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_k)\rangle$ rather than $\langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle$ because $\mathbf{v}_1$ has a much smaller bitsize than $\mathbf{v}_k$, resulting in a better complexity in exact arithmetic. Moreover, in the case where we use floating-point arithmetic, $\mathbf{v}_1$ does not introduce any floating-point error, unlike $\mathbf{v}_k$. Algorithm 3 sums up these enhancements.

Proposition 3.5. *If* $\mathbf{B}$ *is an isometric basis, then Algorithm 3 returns the GSO of* $\mathbf{B}$*. Moreover, if we disregard the computational cost of* r*, then Algorithm 3 performs essentially* $3n^2$ *multiplications (resp. additions), whereas Algorithm 2 performs essentially* $5n^2$ *multiplications (resp. additions).*

Proof. For the correctness of Algorithm 3, one only needs to show that at each step, $C_k = \langle \mathbf{v}_k, r(\tilde{\mathbf{b}}_k)\rangle$ and $D_k = \|\tilde{\mathbf{b}}_k\|^2 = \|\mathbf{v}_k\|^2$. The first and third equalities

Algorithm 3 `Faster_Isometric_GSO`($\mathbf{B}$)

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$
Ensure: Gram-Schmidt reduced basis $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$(, $(C_k)_{1 \leqslant k < n}, (D_k)_{1 \leqslant k < n}$)
1: $\tilde{\mathbf{b}}_1 \leftarrow \mathbf{b}_1$
2: $\mathbf{v}_1 \leftarrow \mathbf{b}_1$
3: $C_1 \leftarrow \langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_1) \rangle$
4: $D_1 \leftarrow \|\mathbf{b}_1\|^2$
5: **for** $k = 1, ..., n-1$ **do**
6: $\quad \tilde{\mathbf{b}}_{k+1} \leftarrow r(\tilde{\mathbf{b}}_k) - \frac{C_k}{D_k} \mathbf{v}_k$
7: $\quad \mathbf{v}_{k+1} \leftarrow \mathbf{v}_k - \frac{C_k}{D_k} r(\tilde{\mathbf{b}}_k)$
8: $\quad C_{k+1} \leftarrow \langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_{k+1}) \rangle$
9: $\quad D_{k+1} \leftarrow D_k - \frac{C_k^2}{D_k}$
10: **end for**

are given by lemma 3.4, and the second one by induction : assuming that C_k, D_k are correct, D_{k+1} is correct, once again from lemma 3.4.

□

4 Gram-Schmidt Decomposition over Isometric Bases

In this section, we show that the computation of the matrix μ from the Gram-Schmidt decomposition (or GSD, see Definition 2.5) can be sped up by a $O(n)$ factor in the case of isometric matrices by using tricks similar to those which led to the speeding-up of GSO. The proof of the following theorem explains how to compute the GSD of an isometric basis/matrix in quadratic time.

Theorem 4.1. *Let $\mathbf{B} = (\mathbf{b}_1, ..., \mathbf{b}_n)$ be an isometric basis and $\tilde{\mathbf{B}} = (\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n)$ its GSO. For the sake of simplicity, we identify the basis $\mathbf{B}$ (resp. $\tilde{\mathbf{B}}$) to the (not necessarily square) matrix which rows are the vectors of the basis. Assume we already have $\mathbf{B}$ and $\tilde{\mathbf{B}}$, along with the values $C_j = \langle \mathbf{v}_j, r(\tilde{\mathbf{b}}_j) \rangle, D_j = \|\tilde{\mathbf{b}}_j\|^2$ for $1 \leqslant j < n$. Then the matrix μ associated to $\mathbf{B}$ can be computed in time $O(n^2)$.*

Proof. For $1 \leqslant i < j \leqslant n$, let $X_{i,j} = \langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle$ and $Y_{i,j} = \langle r(\mathbf{b}_i), \mathbf{v}_j \rangle$. All the nontrivial values of $\mu_{i,j}$ (that is, the values $\mu_{i,j}$ for $1 \leqslant j < i \leqslant n$) can be expressed as $\mu_{i,j} = \frac{X_{i,j}}{D_j}$. The values $X_{i,j}, Y_{i,j}$ satisfy these recursive formulae:

$$\begin{cases} X_{i+1,j+1} = X_{i,j} - \frac{C_j}{D_j} Y_{i,j} \\ Y_{i,j+1} = Y_{i,j} - \frac{C_j}{D_j} X_{i,j} \end{cases}$$

These formulae allow us to compute all the values of $X_{i,j}, Y_{i,j}$ from the $2(n-1)$ values $X_{i,1}, Y_{i,1}$. Once all of these values are computed, one can simply obtain the $\mu_{i,j}$ from the $X_{i,j}$. Algorithm 4 puts this idea into practice.

Algorithm 4 `Isometric_GSD`$(\mathbf{B}, \tilde{\mathbf{B}}, (C_i), (D_i))$

Require: Basis $\mathbf{B}$ and its orthogonalization $\tilde{\mathbf{B}}$, values $C_j = \langle \mathbf{v}_j, r(\tilde{\mathbf{b}}_j)\rangle, D_j = \|\tilde{\mathbf{b}}_j\|^2$ for $1 \leqslant j < n$
Ensure: Matrix $\mu = \mathbf{B} \times \tilde{\mathbf{B}}^{-1}$
1: Set the diagonal values of μ to 1 and the values above the diagonal to 0
2: **for** $i = 2...n$ **do** {Computing the $(X_{i,1}), (Y_{i,1})$}
3: $X_{i,1} \leftarrow \langle \mathbf{b}_i, \tilde{\mathbf{b}}_1 \rangle$
4: $Y_{i,1} \leftarrow \langle r(\mathbf{b}_i), \mathbf{b}_1 \rangle$
5: **for** $j = 2...i-1$ **do**
6: $X_{i,j} \leftarrow X_{i-1,j-1} - \frac{C_{j-1}}{D_{j-1}} Y_{i-1,j-1}$
7: $Y_{i,j} \leftarrow Y_{i,j-1} - \frac{C_{j-1}}{D_{j-1}} X_{i,j-1}$
8: **end for**
9: **end for**
10: **for** $i = 2...n$ **do** {Filling out the non-trivial values of $\mu_{i,j}$}
11: **for** $j = 1...i-1$ **do**
12: $\mu_{i,j} \leftarrow \frac{X_{i,j}}{D_j}$
13: **end for**
14: **end for**

The idea of this algorithm is somewhat similar to the one behind Algorithms 2 and 3: the only values that we really need to compute are the $X_{i,j}$'s, but in order to do that efficiently we resort to a mutual recursion involving the $Y_{i,j}$'s.

The time complexity is straightforward. Each $X_{i,j}, Y_{i,j}$ takes time $O(1)$ to be computed, except for $2n$ of them which need time $O(n)$ each. So the overall cost is $O(n^2)$. □

As an example, the matrices X_{steps} and X_{chrono} below show, for $n = 5$, in which order the matrices X, Y are filled. The two matrices use different metrics: X_{chrono} displays the chronological order in which the matrices are filled by the algorithm, whereas X_{steps} display the minimal depth of the computational tree necessary in order to compute an $X_{i,j}$ (resp. $Y_{i,j}$). If a box contains $\times$, it means that the corresponding value is trivial (see step 1 of the algorithm).

$$X_{steps} = \begin{array}{|c|c|c|c|c|}\hline \times & \times & \times & \times & \times \\ \hline 1 & \times & \times & \times & \times \\ \hline 1 & 2 & \times & \times & \times \\ \hline 1 & 2 & 3 & \times & \times \\ \hline 1 & 2 & 3 & 4 & \times \\ \hline \end{array} \qquad X_{chrono} = \begin{array}{|c|c|c|c|c|}\hline \times & \times & \times & \times & \times \\ \hline 1 & \times & \times & \times & \times \\ \hline 2 & 3 & \times & \times & \times \\ \hline 4 & 5 & 6 & \times & \times \\ \hline 7 & 8 & 9 & 10 & \times \\ \hline \end{array}$$

As X_{chrono} shows, the algorithm fills the matrices X, Y row after row, but if necessary, it could be rewritten in order to fill X, Y column after column, as shown by X_{steps}.

5 Extending the Results to Block Isometric Bases

In previous sections, we showed that we can gain a factor $O(n)$ improvement when performing operations such as Gram-Schmidt decomposition on isometric bases. In this section, we show that these results can be extended to block isometric bases, that is bases that are concatenations of isometric bases.

Definition 5.1. *Let $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_{kn}\}$ be a basis. We say that $\mathbf{B}$ is block isometric if there exist k isometric bases $\mathbf{B}^{(1)}, ..., \mathbf{B}^{(k)}$ such that $\mathbf{B}$ is the concatenation of all these bases.*

The main idea of Algorithm 5 is to use the hypothesis that $r(Span(\mathbf{B}^{(i)})) = Span(\mathbf{B}^{(i)})$ (which in practice is always verified for ideal lattices) in conjunction with part 2 of Proposition 2.3 : if $\tilde{\mathbf{b}}$ is the GSR of $\mathbf{b}$ w.r.t. a block $\mathbf{B}^{(i)}$, then $r(\tilde{\mathbf{b}})$ will be the GSR of $r(\mathbf{b})$ w.r.t. that same block $\mathbf{B}^{(i)}$.

Lemma 5.2. *Assume :*
- *$\mathbf{B}^{(1)}, ..., \mathbf{B}^{(k)}$ are matrices isometric for the same isometry r, and of same rank n*
- *$\forall i \in [\![1, k-1]\!], r(Span(\mathbf{B}^{(i)})) = Span(\mathbf{B}^{(i)})$*

Then Algorithm 5 compute the GSO of $\mathbf{B} = \{\mathbf{B}^{(1)}, ..., \mathbf{B}^{(k)}\} = \{\mathbf{b}_1, ..., \mathbf{b}_{kn}\}$ in $O(k^2nd)$ elementary operations over the scalars.

Algorithm 5 `Block_GSO`$(\mathbf{B})$

Require: Block isometric basis $\mathbf{B} = \{\mathbf{B}^{(1)}, ..., \mathbf{B}^{(k)}\} = \{\mathbf{b}_1, ..., \mathbf{b}_{kn}\}$
Ensure: Gram-Schmidt reduced basis $\tilde{\mathbf{B}}$
1: **for** $i = 0, ..., k-1$ **do**
2: $\quad\tilde{\mathbf{b}}_{ni+1} \leftarrow \mathbf{b}_{ni+1}$
3: $\quad$**for** $j = 1, ..., ni$ **do**
4: $\quad\quad\tilde{\mathbf{b}}_{ni+1} \leftarrow \mathbf{b}_{ni+1} - \frac{\langle \mathbf{b}_{ni+1}, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2} \tilde{\mathbf{b}}_j$ {Make $\tilde{\mathbf{b}}_{ni+1}$ orthogonal to previous vectors}
5: $\quad$**end for**
6: $\quad\tilde{\mathbf{B}}^{(i+1)} \leftarrow \{\mathbf{b}_{ni+1}, r(\mathbf{b}_{ni+1}), ..., r^{n-1}(\mathbf{b}_{ni+1})\}$
7: $\quad\tilde{\mathbf{B}}^{(i+1)} \leftarrow$ `Faster_Isometric_GSO`$(\tilde{\mathbf{B}}^{(i+1)})$
8: **end for**

Proof. We prove correctness by showing inductively that at the end of each iteration i of the outer loop, the $n(i+1)$ first vectors $\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_{n(i+1)}$ are the GSO of $\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_{n(i+1)}$:
- For $i = 0$, this is the case since $\tilde{\mathbf{B}}^{(1)}$ is simply the GSO of $\mathbf{B}^{(i)}$
- If it is verified until step $i-1$, then at step i the vector $\tilde{\mathbf{b}}_{ni+1}$ computed in lines 2-5 of the algorithm is exactly the GSR of $\mathbf{b}_{ni+1}$. Its rotations are orthogonal to the vectors of the previous blocks because r preserves the dot product and $\forall i, r(Span(\mathbf{B}^{(i)})) = Span(\mathbf{B}^{(i)})$, and one can verify that

$\mathbf{b}_{ni+j} - r^{j-1}(\tilde{\mathbf{b}}_{ni+1}) \in Span\{\tilde{\mathbf{B}}^{(1)}...\tilde{\mathbf{B}}^{(i-1)}\}$, so $r^{j-1}(\tilde{\mathbf{b}}_{ni+1})$ is exactly the orthogonalization of $\mathbf{b}_{ni+j}$ w.r.t. $\mathbf{b}_1, ..., \tilde{\mathbf{b}}_{ni}$. The basis computed at line 6 is isometric, so applying `Faster_Isometric_GSO` effectively orthogonalize it.

We now study the complexity of algorithm 5. At each iteration i of the algorithm, the orthogonalization of $\mathbf{b}_{ni+1}$ w.r.t. previous vectors (steps 3 to 5) take time $O(nid)$, and steps 6-7 take time $O(nd)$. So the total complexity is $O(k^2nd)$, gaining a factor n when compared to the complexity $O((kn)^2d)$ of the naive Gram-Schmidt orthogonalisation. □

The GSD can be sped up too. We will not detail it, but Fig. 2 gives the outline on how to use Algorithm 4 on a two-blocks isometric basis.

$$\tilde{\mathbf{B}} = \begin{bmatrix}\mathbf{B}^{(1)}\\ \mathbf{B}^{(2)}\end{bmatrix} \Rightarrow \begin{bmatrix}\tilde{\mathbf{B}}^{(1)}\\ \mathbf{B}^{(2)}\end{bmatrix} \Rightarrow \begin{bmatrix}\tilde{\mathbf{B}}^{(1)}\\ \{\tilde{\mathbf{b}}_{n+1}, ..., r^{n-1}(\tilde{\mathbf{b}}_{n+1})\}\end{bmatrix} \Rightarrow \begin{bmatrix}\tilde{\mathbf{B}}^{(1)}\\ \tilde{\mathbf{B}}^{(2)}\end{bmatrix}$$

$$\mu = \left[\begin{array}{c|c} I_n & 0_n \\ \hline 0_n & I_n \end{array}\right] \Rightarrow \left[\begin{array}{c|c} \mu_1 & 0_n \\ \hline 0_n & I_n \end{array}\right] \Rightarrow \qquad \left[\begin{array}{c|c} \mu_1 & 0_n \\ \hline \mu_3 & I_n \end{array}\right] \qquad \Rightarrow \left[\begin{array}{c|c} \mu_1 & 0_n \\ \hline \mu_3 & \mu_4 \end{array}\right]$$

Fig. 2. Computing the GSD of a two-block isometric basis. $\tilde{\mathbf{B}}$ and μ always satisfy $\mu \times \tilde{\mathbf{B}} = \mathbf{B}$

6 GSO and GSD in Exact Arithmetic

Generally, GSO and GSD are performed over real bases, so the standard way of implementing it is by using floating-point arithmetic. However, this can result in rounding errors: several books and articles discuss this problem with a good introduction being [Hig02].

When the input vectors are in $\mathbb{Z}^d$, as it is very often the case in lattice-based cryptography, then the GSD can be performed using only exact arithmetic over $\mathbb{Q}$. Moreover, some algorithms such as the original LLL algorithm [LLL82] explicitely perform exact GSD.

However, this gain in precision comes at the cost of reduced efficiency: when an integer basis undergoes GSO, the reduced vectors' bitsize quickly escalates in the dimension of the basis and of the underlying space. This phenomenom is called *coefficient explosion* and impacts the space *and* computational cost of GSD. In this section, we adapt Algorithms 3 and 4 to the exact arithmetic setting and show that we still gain a $O(d)$ factor compared to classical GSO/GSD. Moreover, our adapted algorithms completely avoid rational arithmetic.

Through this section, we make an additional "niceness" assumption over the isometry r, namely we suppose that it maps integer vectors into integer vectors: $\forall \mathbf{b} \in \mathbb{Z}^d, r(\mathbf{b}) \in \mathbb{Z}^d$.

6.1 GSO in Exact Arithmetic

Definition 6.1. *Let $\mathbf{B} = (\mathbf{b}_j)_{1 \leqslant j \leqslant n}$ be an isometric basis, and for $j \in [\![1, n]\!]$, $\tilde{\mathbf{b}}_j, \mathbf{v}_j, C_j, D_j$ be defined as in Section 3. We then define, $\forall i, j \in [\![1, n]\!]$, the following values:*

- $\lambda_{j,j} = \prod_{1 \leqslant k \leqslant j} \|\tilde{\mathbf{b}}_k\|^2$
- $\tilde{\mathbf{db}}_j = \lambda_{j-1,j-1}\tilde{\mathbf{b}}_j$
- $c_j = \lambda_{j-1,j-1}C_j$
- $\lambda_{i,j} = \mu_{i,j}\lambda_{j,j}$
- $\mathbf{dv}_j = \lambda_{j-1,j-1}\mathbf{v}_j$
- $d_j = \lambda_{j-1,j-1}D_j$

Proposition 6.2. *Using notations of Definition 6.1, $\forall i, j \in [\![1, n]\!]$, we have:*

1. $\lambda_{i,j} \in \mathbb{Z}$
2. $\tilde{\mathbf{db}}_j, \mathbf{dv}_j \in \mathbb{Z}^d$
3. $c_j, d_j \in \mathbb{Z}$

Proof. Proofs for assertions 1 and 2 can be found per example in [Gal12, chapter 17, theorem 17.3.2]. As for assertion 3, $d_j = \lambda_{j,j}$ and $c_j = \langle \mathbf{v}_1, r(\tilde{\mathbf{db}}_j)\rangle$, where $\mathbf{v}_1$ and $r(\tilde{\mathbf{db}}_j)$ are in $\mathbb{Z}^d$. □

With these results in hand, we can now devise an integer version of Algorithm 3. Instead of outputting rational values, Algorithm 6 outputs only integers and integer vectors, and one can then retrieve any vector $\tilde{\mathbf{b}}_k$ by computing $\tilde{\mathbf{b}}_k = \frac{1}{\sqrt{d_{k-1}}}\tilde{\mathbf{db}}_k$. Algorithm 6 uses no rational number and all the internal operations, including exact divisions in steps 6,7 and 9, output integer values. The following lemma shows that in the case we use exact arithmetic, Algorithm 6 is still at least $O(n)$ faster than standard GSO.

Algorithm 6 `Integer_Isometric_GSO`$(\mathbf{B})$

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$
Ensure: $(\tilde{\mathbf{db}}_k, \mathbf{dv}_k, c_k, d_k)_{k=1...n}$ as defined in Definition 6.1

1: $\tilde{\mathbf{db}}_1 \leftarrow \mathbf{b}_1$
2: $\mathbf{dv}_1 \leftarrow \mathbf{b}_1$
3: $c_1 \leftarrow \langle r(\tilde{\mathbf{b}}_1), \mathbf{dv}_1\rangle$
4: $d_1 \leftarrow \|\mathbf{b}_1\|^2$
5: **for** $k = 1, ..., n-1$ **do**
6: $\quad \tilde{\mathbf{db}}_{k+1} \leftarrow \left[d_k r(\tilde{\mathbf{db}}_k) - c_k \mathbf{dv}_k\right] / d_{k-1}$
7: $\quad \mathbf{dv}_{k+1} \leftarrow \left[d_k \mathbf{dv}_k - c_k r(\tilde{\mathbf{db}}_k)\right] / d_{k-1}$
8: $\quad c_{k+1} \leftarrow \langle \mathbf{v}_1, r(\tilde{\mathbf{db}}_{k+1})\rangle$
9: $\quad d_{k+1} \leftarrow \frac{d_k^2 - c_k^2}{d_{k-1}}$
10: **end for**

Lemma 6.3. *Let* $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\} \in (\mathbb{Z}^d)^n$ *be an integral isometric basis,* $|\mathbf{B}| = \max_{k=1...n} (\|\mathbf{b}_k\|)$ *and* $\mathcal{M}(X)$ *denote the time complexity for multiplying two integers of at most* X *bits. Suppose the isometry* r *associated to* $\mathbf{B}$ *can be computed in time and space linear to the size of the input. Then Algorithm 6 performs in time* $O(dn\mathcal{M}(n \log |\mathbf{B}|))$.

Proof. By definition, $d_k = \prod_{1 \leqslant i \leqslant k} \|\tilde{\mathbf{b}}_i\|^2$ so $|d_k| \leqslant |\mathbf{B}|^{2k}$. Moreover, $|C_k| < D_k$ implies $|c_k| < d_k$ and therefore c_k, d_k both have bitsizes $O(k \log |\mathbf{B}|)$. On the other hand, $\tilde{\mathbf{db}}_k$ (resp. $\mathbf{dv}_k$) has its norm less than $|\mathbf{B}|^{2k-1}$ so the four scalar-vectors products performed on steps 6,7 have complexity $O(d\mathcal{M}(k \log |\mathbf{B}|))$, as well as the two divisions of vectors by scalars (we recall that euclidean division of X bit numbers can be performed in time $O(\mathcal{M}(X))$). Overall, each iteration k of the **for** loop takes time $O(d\mathcal{M}(k \log |\mathbf{B}|))$, so the total complexity of Algorithm 6 is $O(dn\mathcal{M}(n \log |\mathbf{B}|))$.

□

6.2 GSD in Exact Arithmetic

The isometric GSD can also naturally be converted into an efficient, "rational-free" version. Let $x_{i,j} \stackrel{\Delta}{=} \lambda_{j-1} X_{i,j} = \langle \mathbf{b}_i, \tilde{\mathbf{db}}_j \rangle \in \mathbb{Z}$ and $y_{i,j} \stackrel{\Delta}{=} \lambda_{j-1} Y_{i,j} = \langle r(\mathbf{b}_i), \mathbf{dv}_j \rangle \in \mathbb{Z}$. The relations

$$\begin{cases} X_{i+1,j+1} = X_{i,j} - \frac{C_j}{D_j} Y_{i,j} \\ Y_{i,j+1} = Y_{i,j} - \frac{C_j}{D_j} X_{i,j} \end{cases}$$

then become

$$\begin{cases} x_{i+1,j+1} = \frac{d_j x_{i,j} - c_j y_{i,j}}{d_{j-1}} \\ y_{i,j+1} = \frac{d_j y_{i,j} - c_j x_{i,j}}{d_{j-1}} \end{cases}$$

The $x_{i,j}$'s actually are the $\lambda_{i,j}$'s, but in Algorithm 7 we continue to write $x_{i,j}$ since it highlights the natural transformation of Algorithm 4 to Algorithm 7.

Algorithm 7 `Integer_Isometric_GSD`$(\mathbf{B}, (c_k, d_k)_{k=1...n})$

Require: Basis $\mathbf{B}$ and the values $(c_k, d_k)_{k=1...n}$
Ensure: $x_{i,j}$'s,$y_{i,j}$'s as defined above
1: **for** $i = 2...n$ **do** {Computing the $(x_{i,1}), (y_{i,1})$}
2: $x_{i,1} \leftarrow \langle \mathbf{b}_i, \tilde{\mathbf{b}}_1 \rangle$
3: $y_{i,1} \leftarrow \langle r(\mathbf{b}_i), \mathbf{b}_1 \rangle$
4: **for** $j = 2...i-1$ **do**
5: $x_{i,j} \leftarrow [d_{j-1} x_{i-1,j-1} - c_{j-1} y_{i-1,j-1}] / d_{j-2}$
6: $y_{i,j} \leftarrow [d_{j-1} y_{i,j-1} - c_{j-1} x_{i,j-1}] / d_{j-2}$
7: **end for**
8: **end for**

Lemma 6.4. *Following the notations of Lemma 6.3, the time complexity of Algorithm 7 is* $O(n^2\mathcal{M}(n\log|\mathbf{B}|) + nd\mathcal{M}(\log|\mathbf{B}|))$.

Proof. The costliest operations of Algorithm 7 are either the $2(n-1)$ dot products in steps 2 and 3, which cost $O(d\mathcal{M}(\log|\mathbf{B}|))$ each, or the essentially $3n^2$ multiplications and divisions made at steps 5 and 6, which cost $O(\mathcal{M}(j\log|\mathbf{B}|))$ each. Summing these costs yields the result. □

Note that in practice d is not much bigger than n, so the complexity of Algorithm 7 becomes $O(n^2\mathcal{M}(n\log|\mathbf{B}|))$. Even in exact arithmetic, our GSO and GSD algorithms still perform $O(n)$ times faster than standard GSD, which complexity is $O(dn^2\mathcal{M}(n\log|\mathbf{B}|))$ (implicit in the proof of [Gal12, Theorem 17.3.4]). Moreover, we manage to avoid the use of any rational number, making our algorithms both efficient and easy to implement.

7 NTRU Lattices

NTRU lattices are a special class of lattices widely used in cryptography, because their *ideal* structure allows a gain of a factor n both in time and space when performing usual operations over lattices. This results in efficient and compact cryptosystems (e.g. [HPS98,LTV12,DDLL13]).

Definition 7.1. *Let* $N, q \in \mathbb{N}^*$ *and* $f, g, F, G \in \mathbb{Z}_N[x]$ *such that* $fG - gF = q \mod (x^N+1)$. *The NTRU lattice generated by* f, g, F, G *is the lattice generated by the rows of the block matrix*

$$\left[\begin{array}{c|c} \mathcal{A}(f) & \mathcal{A}(g) \\ \hline \mathcal{A}(F) & \mathcal{A}(G) \end{array}\right]$$

Where $\mathcal{A}(p)$ *is the* $N \times N$ *matrix which* i*-th row is the coefficients of* $x^{i-1}\cdot p(x) \mod (x^N+1)$.

In [GHN06], Gama et al. considered exact GSD of NTRU bases. They showed that these lattices verify an algebraic property called symplecticity, which allows them to compute the exact GSD faster than with the standard algorithm, using [GHN06, Corollary 1].

But in addition to being q-symplectic, NTRU bases are also block isometric. So we devised an algorithm to compute the exact GSD of a NTRU basis, by combining three strategies:

- use Algorithms 6 and 7 in order to avoid rational arithmetic (as in [Gal12] and [GHN06])
- use the GSO/GSD strategies for isometric bases detailed in Section 5
- use [GHN06, Corollary 1] to compute only one half of the GSO and get the other for free

We compared our exact reduction algorithm with the ones from [GHN06]. It turns out that our algorithm is faster, both theoretically and in practice, despite computing more information: it provides $\tilde{\mathbf{B}}$ and μ, whereas the algorithms in [GHN06] only provide μ. The timings are summarized in Table 1 and the full implementation can be found at `https://github.com/tprest/Fast-GSD`.

Table 1. Timings for Gram-Schmidt over NTRU bases, in seconds. The implementation was done on `Sage 5.3`. Timings were performed on an Intel Core i5-3210M laptop with a 2.5GHz CPU and 6GB RAM. Isometric GSD is "standard" GSD for block isometric bases, whereas Iso.+Symp. GSD takes into account the observations from [GHN06].

Dimension $n = 2N$	128	256	512	1024
Standard GS [GHN06]	3.22	30.7	390	4536
Dual GS [GHN06]	2.39	17	214	2496
Symplectic GS [GHN06]	0.89	5.73	33.9	279
Isometric GSD	0.48	2.05	12.4	89
Iso.+Symp. GSD	0.312	1.4	8.18	57.8

8 Reversibility and Application to Linear-Storage Gaussian Sampling

Gaussian Sampling [Kle00,GPV08] is a cornerstone of lattice cryptography. It can either serve to find approximately close lattice points close to a vector [Kle00], or to sample a lattice point close to a target point without leaking any information about the basis used [GPV08]. We recall the definition of the Gaussian Sampler:

A drawback of applying the Gaussian Sampler over ideal lattices is that, even though the basis $\mathbf{B}$ of an ideal lattice can be stored using $O(1)$ vectors, this is not the case for the reduced basis $\tilde{\mathbf{B}}$, which needs n vectors. This can quickly impede the practicality of the Gaussian Sampler: for example, for $n = 1024$ (a typical dimension for cryptographic lattices), if $\tilde{\mathbf{B}}$ is stored using 128 bits of precision, the bitsize of $\tilde{\mathbf{B}}$ then exceeds 128 Mbits.

Our algorithm allows to overcome this problem by computing the reduced basis $\tilde{\mathbf{B}}$ on-the-fly. An obstacle is that Gaussian Sampling needs the vectors of $\tilde{\mathbf{B}}$ *in reverse order*, so a straightforward use of Algorithm 2 or 3 does not solve the problem since it provides the basis in direct order. Fortunately, as Fig. 1 suggests, Algorithms 2 and 3 can be "reversed" in the sense that provided with the last vector $\tilde{\mathbf{b}}_n$ of the basis $\tilde{\mathbf{B}}$ and a few extra pieces of information, one can compute $\tilde{\mathbf{b}}_{n-1}, ..., \tilde{\mathbf{b}}_1$ on-the-fly.

Algorithm 8 `Gaussian_Sampler`$(\mathbf{B}, \tilde{\mathbf{B}}, \sigma, \mathbf{c})$

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$, its GSO $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$, $\sigma > 0$, center $\mathbf{c} \in \mathbb{Z}^m$
Ensure: $\mathbf{z}$ sampled in $\mathcal{D}_{\Lambda(\mathbf{B}),\sigma,\mathbf{c}}$

$\mathbf{c}_n \leftarrow \mathbf{c}$
for $i \leftarrow n, ..., 1$ **do**
  $d_i \leftarrow \langle \mathbf{c}_i, \tilde{\mathbf{b}}_i \rangle / \|\tilde{\mathbf{b}}_i\|^2$
  $\sigma_i \leftarrow \sigma / \|\tilde{\mathbf{b}}_i\|$
  $z_i \leftarrow D_{\mathbb{Z},\sigma_i,d_i}$
  $\mathbf{c}_{i-1} \leftarrow \mathbf{c}_i - z_i \boldsymbol{b}_i$
end for
return $\mathbf{c} - \mathbf{c}_0$

Definition 8.1. *For a basis* $\mathbf{B}$*, we denote, for any* $i \in [\![1; n-1]\!]$*,* $C_i = \langle \mathbf{v}_i, r(\tilde{\mathbf{b}}_i) \rangle$ *and* $D_i = \|\tilde{\mathbf{b}}_i\|^2$*. We also define* $H_i = \frac{1}{1-(C_i/D_i)^2} = \frac{D_i}{D_{i+1}}$ *and* $I_i = \frac{C_i/D_i}{1-(C_i/D_i)^2} = \frac{C_i}{D_{i+1}}$*.*

Algorithm 9 `Compact_Gaussian_Sampler`$(\mathbf{B}, \tilde{\mathbf{B}}, \sigma, \mathbf{c}, \tilde{\mathbf{b}}_n, \mathbf{v}_n, \mathbf{H}, \mathbf{I})$

Require: Basis $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$, center $\mathbf{c} \in \mathbb{Z}^m$, precomputed vectors $\tilde{\mathbf{b}}_n, \mathbf{v}_n$, precomputed values $(H_i, I_i)_{1 \leqslant i < n}$ from definition 8.1
Ensure: $\mathbf{z}$ sampled in $\mathcal{D}_{\Lambda(\mathbf{B}),\sigma,\mathbf{c}}$

$\mathbf{c}_n \leftarrow \mathbf{c}$
for $i \leftarrow n, ..., 1$ **do**
  $d_i \leftarrow \langle \mathbf{c}_i, \tilde{\mathbf{b}}_i \rangle / \|\tilde{\mathbf{b}}_i\|^2$
  $\sigma_i \leftarrow \sigma / \|\tilde{\mathbf{b}}_i\|$
  $z_i \leftarrow D_{\mathbb{Z},\sigma_i,d_i}$
  $\mathbf{c}_{i-1} \leftarrow \mathbf{c}_i - z_i \mathbf{b}_i$
  $\tilde{\mathbf{b}}_{i-1} \leftarrow r^{-1}(H_{i-1}\tilde{\mathbf{b}}_i + I_{i-1}\mathbf{v}_i)$
  $\mathbf{v}_{i-1} \leftarrow I_{i-1}\tilde{\mathbf{b}}_i + H_{i-1}\mathbf{v}_i$
end for
return $\mathbf{c} - \mathbf{c}_0$

Lemma 8.2. *Algorithms 8 and 9 produce the same output when they have the same input* $\mathbf{B}$ *and* $\mathbf{c}$ *(assuming the associated precomputed values are correct).*

Proof. First, observe that $\forall i = 1...n-1$, $|C_i| < D_i$, because otherwise D_{i+1} would be zero and $\mathbf{B}$ would not be a basis. One can see that the "linear system"

- $\tilde{\mathbf{b}}_{i+1} = r(\tilde{\mathbf{b}}_i) - \frac{C_i}{D_i}\mathbf{v}_i$
- $\mathbf{v}_{i+1} = \mathbf{v}_i - \frac{C_i}{D_i} r(\tilde{\mathbf{b}}_i)$

is invertible :
- $\tilde{\mathbf{b}}_i = r^{-1}(H_i\tilde{\mathbf{b}}_{i+1} + I_i\mathbf{v}_{i+1})$
- $\mathbf{v}_i = I_i\tilde{\mathbf{b}}_{i+1} + H_i\mathbf{v}_{i+1}$

H_i and I_i are always defined since $|C_i| < D_i$. Therefore, the same way the values C_i, D_i allow to compute $\tilde{\mathbf{b}}_{i+1}, \mathbf{v}_{i+1}$ from $\tilde{\mathbf{b}}_i, \mathbf{v}_i$, H_i, I_i allow to compute $\tilde{\mathbf{b}}_i, \mathbf{v}_i$ from $\tilde{\mathbf{b}}_{i+1}, \mathbf{v}_{i+1}$. □

This allows us to perform Gaussian Sampling using $O(m)$ memory space instead of $O(mn)$ for the classic version. The overhead in time is reasonable :
- Classic Sampler: $2mn$ additions, $2mn$ multiplications, n samplings in $\mathbb{Z}$
- Compact Sampler: $4mn$ additions , $6mn$ multiplications, n samplings in $\mathbb{Z}$

Therefore, the compact Gaussian Sampler is *at most* three times slower than the classic one. This is confirmed by experiments summarised in Table **??**. Moreover, in Algorithm 9, it is possible to sample around several $\mathbf{c}$'s at the same time: this then makes negligible the overhead induced by the addition (in Algorithm 9) of lines 7 and 8. This time-memory trade-off allows to do Gaussian Sampling for k targets in space $O(km)$ and in time *at most* $\left(1 + \frac{2}{k}\right)$ times the time required by the classic Gaussian Sampler.

8.1 Analysis of the Space Requirement for the Gaussian Sampler

Suppose that $\tilde{\mathbf{B}}$ needs to be known up to $|\log_2 \epsilon|$ bits, for some $\epsilon < 1$. In order to be able to run Algorithm 9 any time (without having to undergo the GSO beforehand), one only needs to store $(H_i, I_i, \|\tilde{\mathbf{b}}_i\|)_{i=1\dots n}$ as well as $\tilde{\mathbf{b}}_n, \mathbf{v}_n$. However, it is straightforward to use the relation $\frac{\|\tilde{\mathbf{b}}_{i+1}\|^2}{\|\tilde{\mathbf{b}}_i\|^2} = 1 - \left(\frac{C_i}{D_i}\right)^2$ to save even more space by just storing the $\|\tilde{\mathbf{b}}_i\|$'s and deriving the H_i's, I_i's from them. During the execution of Algorithm 9, one also needs to store the current $\tilde{\mathbf{b}}_i, \mathbf{v}_i$. So overall the space requirement of Algorithm 9 is $5n(|\log_2 \epsilon| + b)$, where b is less the "number of bits lost" in steps 6, 7 of Algorithm 9: in other words, b is such that if $\tilde{\mathbf{b}}_n, \mathbf{v}_n, (\|\tilde{\mathbf{b}}_i\|)_{i=1\dots n}$ are known up to $|\log_2 \epsilon| + b$ bits, then $\tilde{\mathbf{B}}$ is guaranteed to be known up to $|\log_2 \epsilon|$ bits.

For NTRU lattices, this analysis can be refined: only half of the $\|\tilde{\mathbf{b}}_i\|$ need to be known, and $\tilde{\mathbf{b}}_n, \mathbf{v}_n$ can be determined from $\mathbf{b}_1 = \tilde{\mathbf{b}}_1$ [GHN06, Corollary 1]. Instead of needing to know $3n(|\log_2 \epsilon| + b)$ bits beforehand, we just need $\frac{n}{2}(|\log_2 \epsilon| + b)$, so the total space requirement is $2.5n(|\log_2 \epsilon| + b)$.

9 Precision of the Gaussian Sampler

It is known [GPV08] that for σ big enough, the output f of Algorithm 8 is statistically close to the distribution $\mathcal{D}_{\Lambda(\mathbf{B}),\sigma,\mathbf{c}}$. However, the proof holds only when $\mathbf{B}, \tilde{\mathbf{B}}, \sigma, \mathbf{c}$ and the values $\|\tilde{\mathbf{b}}_i\|$'s are known exactly. But in practice, one can not afford to do computations with the exact representation of $\tilde{\mathbf{B}}$ and of the $\|\tilde{\mathbf{b}}_i\|$'s, as it would be too costly in terms of space and computational resources. Therefore, $\tilde{\mathbf{B}}$ and the $\|\tilde{\mathbf{b}}_i\|$'s are stored up to some finite precision, and this

finite precision introduces errors ϵ, δ_1 which impact the output distribution of the algorithm. Theorem 9.1 bounds the statistical distance $\Delta(f, f_{\epsilon,\delta_1})$ between the output distribution f of the "perfect" algorithm, and the output distribution f_{ϵ,δ_1} of the "imperfect" algorithm.

Theorem 9.1. *Let $m, n, q \in \mathbb{N}^\star$, $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\} \in \mathbb{Z}^{n\times m}$ be a basis of a lattice $\Lambda \subseteq Z^m$, $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$ be the exact GSO of $\mathbf{B}$ and $\mathbf{c} \in \mathbb{Z}_q^m$. Let $\delta, \epsilon, k > 0$, $\sigma \geqslant \max \|\tilde{\mathbf{b}}_i\|$ and for any $i = 1...n$, let $\sigma_i = \frac{\sigma}{\|\tilde{\mathbf{b}}_i\|}$. Let f_{ϵ,δ_1} be the output distribution of Algorithm 8 ran on input $(\mathbf{B}, \tilde{\mathbf{B}}, \sigma, (\|\tilde{\mathbf{b}}_i\|)_i, \mathbf{c})$, where the coefficients of $\tilde{\mathbf{B}}$ are known with absolute precision at least δ_1, and the values $\|\tilde{\mathbf{b}}_i\|$ are known with relative precision at least ϵ. When $\tilde{\mathbf{B}}$ and the $\|\tilde{\mathbf{b}}_i\|$ are known exactly, we simply refer to the output distribution as f.*
Let $\delta_3 = \frac{2k\epsilon q\sqrt{m}}{\sigma} + \epsilon k^2 + \frac{mq\delta_1 k}{\sigma \min_i \|\tilde{\mathbf{b}}_i\|}$. If $\delta_3 \leqslant 1/2$, then:

$$\Delta(f, f_{\epsilon,\delta_1}) \leqslant 2n(\delta_3 + 3e^{-k^2/2})$$

In particular, if we want $\Delta(f, f_{\epsilon,\delta_1})$ to be less than $2^{-\lambda}$ for some $\lambda > 0$, it is enough that:

$$\delta_1, \epsilon \leqslant \frac{2^{-\lambda}}{2n\sqrt{\lambda + \log_2 n}\left(\sqrt{\lambda + \log_2 n} + \frac{2q\sqrt{m}}{\sigma} + \frac{mq}{\sigma \min_i \|\tilde{\mathbf{b}}_i\|}\right)}$$

Proof. f is the output distribution of Algorithm 8 executed on exact input $(\mathbf{B}, \tilde{\mathbf{B}}, \sigma, (\|\tilde{\mathbf{b}}_i\|)_i, \mathbf{c})$, and f_{ϵ,δ_1} is the output distribution of Algorithm 8 executed on input $(\mathbf{B}, \tilde{\mathbf{B}}', \sigma, (\|\tilde{\mathbf{b}}_i\|')_i, \mathbf{c})$, where:

- each vector $\tilde{\mathbf{b}}_i'$ of $\tilde{\mathbf{B}}'$ is correct up to $\log_2 \delta_1$ bits after the comma: $\forall i, \|\tilde{\mathbf{b}}_i - \tilde{\mathbf{b}}_i'\|_\infty \leqslant \delta_1$
- each $\|\mathbf{b}_i\|'$ is correct up to $\log_2 \epsilon$ bits: $\left|\frac{\|\tilde{\mathbf{b}}_i\|}{\|\mathbf{b}_i\|'} - 1\right| \leqslant \epsilon$
- since $\sigma_i = \frac{\sigma}{\|\tilde{\mathbf{b}}_i\|}, \sigma_i' = \frac{\sigma}{\|\tilde{\mathbf{b}}_i\|'}$, each σ_i' is also correct up to $\log_2 \epsilon$ bits

Let $g = f_{\epsilon,\delta_1}$. Out goal is make f and g fall into the conditions of Lemma A.1 for some X, X', γ, δ_5, so that we can get a bound on their statistical distance.
Let $X = \{\sum_{i=1...n} z_i\mathbf{b}_i | (z_1, ..., z_n) \in \mathbb{Z}^n\}$. Each possible output $\mathbf{z} = \sum_i z_i\mathbf{b}_i$ implicitely defines $(d_1, ...d_n)$. Let $X' = \{\mathbf{z} = \sum_i z_i\mathbf{b}_i \big| \forall i, |z_i - d_i| \leqslant k\sigma_i\}$. From [Lyu12, Lemma 4.4, part 1], $f(X\backslash X'), g(X\backslash X') \leqslant 1-(1-4e^{-k^2/2})^n \leqslant 4ne^{-k^2/2}$. On the other hand, Lemma A.5 tells us that for $\delta_4 \approx 4\delta_3 + 4e^{-k^2/2}$, $\forall \mathbf{z} = \sum_{i=1...n} z_i\mathbf{b}_i \in X$ and $\forall i = 1...n$:

$$1 - \delta_4 \leqslant \frac{D_{\mathbb{Z},\sigma_i',d_i'}(z_i)}{D_{\mathbb{Z},\sigma_i,d_i}(z_i)} \leqslant 1 + \delta_4$$

Combining this with $f(\mathbf{z}) = \prod_{i=1...n} D_{\mathbb{Z},\sigma_i,d_i}(z)$ and $g(\mathbf{z}) = \prod_{i=1...n} D_{\mathbb{Z},\sigma_i',d_i'}(z)$, we have a bounded ratio $\frac{g}{f}$ over X':

$$\forall \mathbf{z} \in X', 1 - n\delta_4 \leqslant (1-\delta_4)^n \leqslant \frac{g(\mathbf{z})}{f(\mathbf{z})} \leqslant (1+\delta_4)^n = 1 + n\delta_4 + O(\delta_4^2)$$

Taking $\gamma = 4ne^{-k^2/2}$ and $\delta_5 = n\delta_4$, we can now apply Lemma A.1:

$$2\Delta(f, g) \leqslant n\delta_4 + 2\gamma$$

□

The proof of Theorem 9.1 resorts to several lemmas that can be found in Appendix A.

9.1 Application to NTRU Lattices

We use the formula obtained in Theorem 9.1 in order to derive concrete bounds in the case of NTRU lattices. In this particular case, $m = n, \sigma \min \|\tilde{\mathbf{b}}_i\| > q$ [GHN06, Corollary 1] and $\sigma > \sqrt{q}$ (because $\prod_i \|\tilde{\mathbf{b}}_i\| = q^{n/2}$). We can also reasonably assume that $\log_2 n < \lambda < qn/2$, so

$$\epsilon, \delta_1 < \frac{2^{-\lambda}}{8n\sqrt{\lambda qn}} \qquad \Longrightarrow \qquad \Delta(f, f_{\epsilon,\delta_1}) \leqslant 2^{-\lambda}$$

As an example, for $n = 1024, q \leqslant 2^{26}$ and $\lambda \leqslant 256$, this means that $\Delta(f, f_{\epsilon,\delta_1}) \leqslant 2^{-128}$ provided that $\tilde{\mathbf{B}}$ (resp. the $\|\tilde{\mathbf{b}}_i\|$'s) is known up to $\lambda + 35 + \log_2(\max_i \|\tilde{\mathbf{b}}_i\|)$ (resp. $\lambda + 35$) bits of precision.

We now use the results from Subsection 8.1 to determine the space requirements for the parameters as above. Algorithm 9 requires $2.5n(\lambda + 35 + b)$ bits of space. In experiments we launched on NTRU lattices, we always get $b \leqslant 30$. We will therefore take this value for b.

Table 2. Timings (in milliseconds) and space requirements (in bits and mega-bits) of the classic and compact Gaussian Samplers (Classic GS and Compact GS). The implementation was done in `C++` using `GMP`. Timings were performed on an Intel Core i5-3210M laptop with a 2.5GHz CPU and 6GB RAM.

Statistical distance from ideal		2^{-80}	2^{-128}	2^{-192}
Precision needed	Classic GS	115 bits	163 bits	223 bits
	Compact GS	145 bits	193 bits	253 bits
Running time	Classic GS	115 ms	170 ms	203 ms
	Compact GS	446 ms	521 ms	523 ms
Space requirements	Classic GS	115 Mb	163 Mb	223 Mb
	Compact GS	0.35 Mb	0.47 Mb	0.63 Mb

As a test for the practicality of our compact Gaussian Sampler, we implemented both the classic and compact Gaussian Samplers and compared their

timings and space requirements. As predicted by our computations, the compact Gaussian Sampler is no more than thrice slower than the classic one, while having space requirements smaller by between two and three orders of magnitude. Our results are summarized in Table 2 and the complete implementation can be found on `https://github.com/tprest/Compact-Sampler`.

Acknowledgements

The authors wish to thank Phong Q. Nguyen, as well as the anonymous Eurocrypt'15 reviewers, for helpful comments which helped improve the presentation of this work.

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 98–115, 2010.

[Bab86] László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 533–556, 2014.

[Boy10] Xavier Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In *Public Key Cryptography*, pages 499–517, 2010.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552, 2010.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *CRYPTO (1)*, pages 40–56, 2013.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, pages 22–41, 2014.

[Gal12] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, New York, NY, USA, 1st edition, 2012.

[GHN06] Nicolas Gama, Nick Howgrave-Graham, and Phong Q. Nguyen. Symplectic lattice reduction and ntru. In *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, pages 233–253, Berlin, Heidelberg, 2006. Springer-Verlag.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[Hig02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2002.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In *SODA*, pages 937–941, 2000.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP (2)*, pages 144–155, 2006.

[LMPR08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In *FSE*, pages 54–72, 2008.

[LPR13a] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013. Preliminary version appeared in EUROCRYPT 2010.

[LPR13b] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *EUROCRYPT*, pages 35–54, 2013.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, pages 738–755, 2012.

[MS07] Vladimir Maz'ya and Gunther Schmidt. *Approximate Approximations*. AMS, 1st edition, 2007.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC*, pages 145–166, 2006.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, pages 617–635, 2009.

[Swe84] D.R. Sweet. Fast toeplitz orthogonalization. *Numerische Mathematik*, 43:1–21, 1984.

A Lemmas used in the Precision Analysis of the Gaussian Sampler

This section regroups the lemmas used by Theorem 9.1 in order to bound the statistical distance. We will sometimes resort to approximations such as $\rho_{d,\sigma}(\mathbb{Z}) \approx 1$ in order to simplify computations: indeed, $|\rho_{d',\sigma}(\mathbb{Z}) - 1| < 1.04 \cdot 10^{-8\sigma^2}$ whenever $\sigma > \frac{1}{\sqrt{2}}$ (see e.g. [MS07, Sect. 1.1]), and that will always be the case through this section. In the same way, we always assume ϵ and δ_i's to be very small and will therefore discard δ terms whenever possible. Each time such an approximation is done, it is indicated with signs such as $O(\cdot)$ or $\approx$, and has a negligible impact. More precisely, it never adds "hidden errors" to the result being proven.

The first lemma gives a simple bound on the statistical distance between two distributions f and g which are both in a set X' with probability $1-\gamma$, and enjoy a relative error bound $1-\delta_5 \leqslant \frac{g(\mathbf{z})}{f(\mathbf{z})} \leqslant 1+\delta_5$ over this set X'. As one could expect, the statistical distance between f and g becomes linear in $\delta_5+\gamma$ when $\delta_5, \gamma \to 0$.

Lemma A.1. *Let f, g be two distributions over a set X. Let $X' \subseteq X$, $\delta_5, \gamma > 0$ such that $\sum_{\mathbf{z}\in X\setminus X'} f(\mathbf{z}), \sum_{\mathbf{z}\in X\setminus X'} g(\mathbf{z}) \leqslant \gamma$, and $\forall \mathbf{z} \in X', 1-\delta_5 \leqslant \frac{g(\mathbf{z})}{f(\mathbf{z})} \leqslant 1+\delta_5$. Then:*

$$2\Delta(f,g) \leqslant 2\gamma + \delta_5.$$

Proof. We separate the statistical distance sum into two sums over X' and $X\setminus X'$:

$$\begin{aligned} 2\Delta(f,g) &\leqslant \textstyle\sum_{\mathbf{z}\in X\setminus X'} f(\mathbf{z}) + \sum_{\mathbf{z}\in X\setminus X'} g(\mathbf{z}) + \sum_{\mathbf{z}\in X'} |f(\mathbf{z}) - g(\mathbf{z})| \\ &\leqslant 2\gamma + \delta_5 \sum_{\mathbf{z}\in X'} f(\mathbf{z}) \\ &\leqslant 2\gamma + \delta_5 \end{aligned}$$

□

The following lemma bounds the error occurring in step 3 of Algorithm 8, when the center d_i is computed. In the floating-point version, the $\tilde{\mathbf{b}}_i$ are known up to absolute precision δ_1 (*ie* $\log_2 \delta_1$ bits after the comma) and their norms $\|\tilde{\mathbf{b}}_i\|$ are known up to relative precision ϵ (*ie* $\log_2 \epsilon - \log_2 \|\tilde{\mathbf{b}}_i\|$ bits after the comma), where δ_1 and ϵ are not necessarily equal.

Lemma A.2. *Let $0 < \delta_1, \epsilon \ll 1$, $\mathbf{c} \in \mathbb{Z}_q^m, \mathbf{b}, \mathbf{b}' \in \mathbb{R}^m$ such that $\|\mathbf{b}-\mathbf{b}'\|_\infty \leqslant \delta_1$ and $\left|\frac{\|\mathbf{b}\|}{\|\mathbf{b}'\|} - 1\right| \leqslant \epsilon$. Let $d = \frac{\langle \mathbf{c},\mathbf{b}\rangle}{\|\mathbf{b}\|^2}$, $d' = \frac{\langle \mathbf{c},\mathbf{b}'\rangle}{\|\mathbf{b}'\|^2}$. Then $|d-d'| \leqslant \delta_2$, where $\delta_2 \approx 2\epsilon\frac{q\sqrt{m}}{\|\mathbf{b}\|} + \frac{mq\delta_1}{\|\mathbf{b}\|^2}$.*

Proof. $d' = \left(\frac{\langle \mathbf{c},\mathbf{b}\rangle}{\|\mathbf{b}\|^2} + \frac{\langle \mathbf{c},\mathbf{b}'-\mathbf{b}\rangle}{\|\mathbf{b}\|^2}\right)\frac{\|\mathbf{b}\|^2}{\|\mathbf{b}'\|^2}$, so

$$\begin{aligned} |d'-d| &\leqslant ((1+\epsilon)^2-1)d + \tfrac{\|\mathbf{c}\|_1\|\mathbf{b}'-\mathbf{b}\|_\infty}{\|\mathbf{b}\|^2}(1+\epsilon)^2 \\ &\leqslant 2\tfrac{q\sqrt{m}}{\|\mathbf{b}\|}(\epsilon + O(\epsilon^2)) + \tfrac{mq}{\|\mathbf{b}\|^2}(\delta_1 + O(\delta_1^2)) \end{aligned}$$

□

In the three next lemmas, we study the difference of behaviour between a perfect gaussian over $\mathbb{Z}$ of center d and standard deviation σ, and the same gaussian with a slightly perturbed center d' and standard deviation σ'. For any center d, $D_{\mathbb{Z},\sigma,d}$ can be exactly simulated from $D_{\mathbb{Z},\sigma,d-1}$ (and reciprocally), so we can suppose w.l.o.g. that $d \in (-1/2, 1/2]$. The Lemmas A.3, A.4 progressively build up to establish in Lemma A.5 a bound over the ratio of $D_{\mathbb{Z},\sigma,d}$ and $D_{\mathbb{Z},\sigma',d'}$, which are the distributions from which Algorithm 8 samples in step 5 ($D_{\mathbb{Z},\sigma,d}$ in the "perfect" algorithm, $D_{\mathbb{Z},\sigma',d'}$ in the "imperfect" one).

Lemma A.3. *Let $\epsilon, \delta_2, k > 0, \sigma, \sigma' > 1$ and $d, d' \in (-1/2, 1/2]$ such that $|d - d'| \leqslant \delta_2$ and $\left|\frac{\sigma}{\sigma'} - 1\right| \leqslant \epsilon$. Let $z \in \mathbb{Z}$ such that $|z - d| \leqslant k\sigma$. Then*

$$e^{-\delta_3} \leqslant \frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d}(z)} \leqslant e^{\delta_3}$$

where $\delta_3 = \frac{\delta_2 k}{\sigma} + \epsilon(k^2+1) + O(\epsilon, \delta_2^2, \epsilon\delta_2)$. In particular, if $\delta_3 \leqslant 1/2$, then $\left|\frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d}(z)} - 1\right| \leqslant 2\delta_3$

Proof.

$$\frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d}(z)} = \frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma',d}(z)} \times \frac{\rho_{\sigma,d'}(z)}{\rho_{\sigma,d}(z)}$$

One one hand,

$$\frac{\rho_{\sigma,d'}(z)}{\rho_{\sigma,d}(z)} = e^{\frac{(d'-d)(2z-d'+d)}{2\sigma^2}}, \text{ and } \left|\frac{(d'-d)(2z-d'+d)}{2\sigma^2}\right| \leqslant \frac{k}{\sigma}(\delta_2 + O(\delta_2^2))$$

On the other hand,

$$\frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d'}(z)} = \frac{\sigma}{\sigma'} \times e^{\frac{(z-d')^2}{2\sigma^2} - \frac{(z-d')^2}{2\sigma'^2}}$$

and

$$\left|\frac{(z-d')^2}{2\sigma^2} - \frac{(z-d')^2}{2\sigma'^2}\right| = \frac{(z-d')^2}{2\sigma^2}\left|1 - \frac{\sigma^2}{\sigma'^2}\right| \leqslant k^2(\epsilon + O(\epsilon^2))$$

Combining both inequalities and using $e^x = 1 + x + O(x^2)$ yield:

$$\begin{aligned}\frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d}(z)} &\leqslant (1 + \tfrac{k}{\sigma}(\delta_2 + O(\delta_2^2))\,(1+\epsilon)\left(1 + k^2(\epsilon + O(\epsilon^2)\right) \\ &\leqslant 1 + \tfrac{\delta_2 k}{\sigma} + \epsilon(k^2+1) + O(\epsilon, \delta_2^2, \epsilon\delta_2)\end{aligned}$$

□

Lemma A.4. *Let $\sigma, \sigma' > 1$ and $d, d' \in (-1/2, 1/2]$. Let $k > 0$ and $Z = \{z \in \mathbb{Z}, |z - d| \leqslant k\sigma\}$. Suppose $\exists \delta_3 \in (0, 1/2), \forall z \in Z$, $\left|\frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d}(z)} - 1\right| \leqslant 2\delta_3$. Then:*

$$\sum_{z\in\mathbb{Z}} |\rho_{\sigma',d'}(z) - \rho_{\sigma,d}(z)| \leqslant 2\delta_3 + 4e^{-k^2/2}$$

Proof. We separate the sum in two sums over Z and $\mathbb{Z}\backslash Z$. For the first sum:

$$\sum_{z\in Z} |\rho_{\sigma',d'}(z) - \rho_{\sigma,d}(z)| \leqslant 2\delta_3 \sum_{z\in Z} |\rho_{\sigma,d}(z)| \leqslant 2\delta_3(1 + 1.01 \cdot 10^{-8}) \approx 2\delta_3$$

Now, for the second sum, Lemma 4.4, part 1, from [Lyu12] states that[8]:

$$\text{For any } k > 0, \mathbb{P}\left[|z - d| > k\sigma; z \leftarrow D_{\mathbb{Z},\sigma,d}\right] \leqslant 2e^{-k^2/2}$$

[8] It is actually stated only for $d = 0$, but the proofs holds $\forall d \in \mathbb{R}$.

Using this lemma, it is straightforward that

$$\begin{aligned}\sum_{z\in\mathbb{Z}\setminus Z}|\rho_{\sigma,d}(z)-\rho_{\sigma,d'}(z)| &\leqslant \sum_{z\in\mathbb{Z}\setminus Z}\rho_{\sigma,d}(z)+\rho_{\sigma,d}(z)\\ &\leqslant 4e^{-k^2/2}\left(\rho_{\sigma,c}(\mathbb{Z})+\rho_{\sigma',c'}(\mathbb{Z})\right)\\ &\leqslant 8e^{-k^2/2}\delta_3(1+1.01\cdot 10^{-8})\approx 8e^{-k^2/2}\end{aligned}$$

□

Lemma A.5 (Bounded ratio of discrete gaussians over a finite set). *Let $\sigma,\sigma'>1$ and $d,d'\in(-1/2,1/2]$. Let $k>0$ and $z\in\mathbb{Z}$ such that $|z-d|\leqslant k\sigma$. Suppose $\exists\delta_3\in(0,1/2)$ such that $1-2\delta_3\leqslant\frac{\rho_{\sigma',d'}(z)}{\rho_{\sigma,d}(z)}\leqslant 1+2\delta_3$. Then:*

$$\left|\frac{D_{\mathbb{Z},\sigma',c'}(z)}{D_{\mathbb{Z},\sigma,c}(z)}-1\right|\leqslant\delta_4$$

where $\delta_4=4\delta_3+4e^{-k^2/2}+4\delta_3^2+8\delta_3e^{-k^2/2}$. In practice δ_3 is small and k is "somewhat" big, so $\delta_4\approx 4\delta_3+4e^{-k^2/2}$.

Proof.

$$\begin{aligned}|D_{\mathbb{Z},\sigma,c}(z)-D_{\mathbb{Z},\sigma',c'}(z)| &= \left|\tfrac{\rho_{\sigma,c}(z)}{\rho_{\sigma,c}(\mathbb{Z})}-\tfrac{\rho_{\sigma',c'}(z)}{\rho_{\sigma',c'}(\mathbb{Z})}\right|\\ &= \tfrac{\rho_{\sigma,c}(z)}{\rho_{\sigma,c}(\mathbb{Z})}\left|1-\tfrac{\rho_{\sigma',c'}(z)}{\rho_{\sigma,c}(z)}\times\tfrac{\rho_{\sigma,c}(\mathbb{Z})}{\rho_{\sigma',c'}(\mathbb{Z})}\right|\\ &\leqslant \tfrac{\rho_{\sigma,c}(z)}{\rho_{\sigma,c}(\mathbb{Z})}\left|1-(1+2\delta_3)(1+2\delta_3+4e^{-k^2/2})\right|\\ &\leqslant D_{\mathbb{Z},\sigma,c}(z)\left(4\delta_3+4e^{-k^2/2}+4\delta_3^2+8\delta_3e^{-k^2/2}\right)\end{aligned}$$

where the penultimate line is obtained by bounding $\frac{\rho_{\sigma,c}(\mathbb{Z})}{\rho_{\sigma',c'}(\mathbb{Z})}$ via Lemma A.4.
□