Generating cryptographically-strong random lattice bases and recognizing rotations of $\mathbb{Z}^n$

Tamar Lichter Blanks[*]
Stephen D. Miller[†]
{tl548,miller}@math.rutgers.edu

February 11, 2021

Abstract

Lattice-based cryptography relies on generating random bases which are difficult to fully reduce. Given a lattice basis (such as the private basis for a cryptosystem), all other bases are related by multiplication by matrices in $GL(n,\mathbb{Z})$. How can one sample random elements from $GL(n,\mathbb{Z})$? We consider various methods, finding some are stronger than others with respect to the problem of recognizing rotations of the $\mathbb{Z}^n$ lattice. In particular, the standard algorithm of multiplying unipotent generators together (as implemented in Magma's `RandomSLnZ` command) generates instances of this last problem which can be efficiently broken, even in dimensions nearing 1,500. Similar weaknesses for this problem are found with the random basis generation method in one of the NIST Post-Quantum Cryptography competition submissions (DRS). Other algorithms are described which appear to be much stronger.

keywords: lattices, random basis, integral lattices, unimodular integral matrices, DRS.

[*]Supported by a National Science Foundation Graduate Research Fellowship.
[†]Supported by National Science Foundation Grants CNS-1526333 and CNS-1815562.

1 Introduction

In cryptography one often encounters problems which are easy to solve using a secret private basis of a lattice $\Lambda \subset \mathbb{R}^n$, but are expected to be difficult to solve using suitably-chosen public bases. Famous examples include the Shortest Vector Problem (SVP) and Closest Vector Problem (CVP).

Implicit here is that not only do harder bases exist, but that it is easy to generate them. The main contribution of this paper is to demonstrate that either some lattices lack hard bases altogether, or instead certain basis generation algorithms are cryptographically weak.

In [17] Lenstra and Silverberg posed the challenge of whether highly-symmetric lattices have hard bases, and proved several interesting results along these lines (related to earlier work of Gentry-Szydlo [9]; see also [16, 18]). One particularly beautiful question they posed is:

$$\text{can one efficiently recognize rotations of the standard } \mathbb{Z}^n \text{ lattice?} \tag{1.1}$$

To be more precise, this problem can be stated in two different group-theoretic ways (the second being the formulation in [17, §2]). Let $\{b_1, \ldots, b_n\}$ denote a basis for Λ and let B denote the $n \times n$ matrix whose i-th row is b_i:

Problem 1.

Can one efficiently factor B as $B = MR$, with $M \in GL(n, \mathbb{Z})$ and $R \in O(n)$? (1.2)

Alternatively, following [8] and [17, §2] we may suppose one is given a positive-definite symmetric matrix $G \in SL(n, \mathbb{Z})$ (which we think of as the Gram matrix BB^t of Λ):

Problem 2 ("Recognizing $\mathbb{Z}^n$").

Can one efficiently solve the equation $G = MM^t$ with $M \in GL(n, \mathbb{Z})$? (1.3)

Clearly, Problem 1 reduces to Problem 2 with $G = BB^t$. Conversely, one can orthogonally diagonalize the matrix G in Problem 2 as $G = PDP^t$ for some $P \in O(n)$ and diagonal matrix D with positive diagonal entries. Then $B = PD^{1/2}$ solves the equation $G = BB^t$, and Problem 2 therefore reduces to Problem 1 (modulo technicalities we will not delve into, such as that the entries of P, D, and B will in general be algebraic integers).

Both Problems 1 and 2 have inefficient solutions using sufficiently strong lattice basis reduction. For example, the given information is sufficient to determine whether or not all lattice vector norms are square-roots of integers, and an SVP solver can determine the shortest nonzero norm $\lambda_1(\Lambda)$. If $\lambda_1(\Lambda) \neq 1$, the lattice Λ is definitely not a rotation of $\mathbb{Z}^n$ and Problems 1 and 2 have negative solutions. However, if one finds a vector of norm 1 and all lattice norms are square-roots of integers, it is then easy to see (by subtracting multiples of this vector to obtain an orthogonal complement) that the dimension of the problem reduces from n to $n-1$.

We saw above that finding a non-integral solution $M \in GL(n,\mathbb{R})$ to (1.3) is trivial, since one can orthogonally diagonalize G and take the square-root of a positive diagonal matrix. However, imposing the constraint that $M \in GL(n,\mathbb{Z})$ adds an intricate dose of number theory, since (1.3) then becomes a class number problem: indeed, in large dimensions n there is a combinatorial explosion of possible $GL(n,\mathbb{Z})$-equivalence classes.[1] For that reason we split Problem 2 into two parts:

Problem 2a (Decision version).

Efficiently determine whether or not a solution $M \in GL(n,\mathbb{Z})$ exists to (1.3).

Problem 2b (Search version).

If so, efficiently find such a solution $M \in GL(n,\mathbb{Z})$. (1.4)

It was recently shown in [12] that Problem 2a is in the class NP$\cap$ co-NP, using results of Elkies [5] on characteristic vectors of lattices (see also [10, §9.6]).

This paper primarily concerns Problem 2b, i.e., one is handed a matrix of the form MM^t and wishes to efficiently recover M. Of course permuting the columns of M does not change MM^t, nor does multiplying any subset of columns by -1; thus we look for solutions up to such signed permutations of the columns. (For this reason it is equivalent to insist that $M \in SL(n,\mathbb{Z})$.) We find that the choice of procedure to randomly generate instances of M has

[1]For example, the E_8 lattice has a Gram matrix G in $SL(8,\mathbb{Z})$, but is not equivalent to the $\mathbb{Z}^8$ lattice. In general the number of $GL(n,\mathbb{Z})$-equivalence classes of such integral unimodular lattices grows faster than exponentially in n [3, Chapter 16].

a drastic impact on the difficulty of the problem. We state this in terms of a probability density function $p : GL(n, \mathbb{Z}) \to \mathbb{R}_{\geq 0}$ (i.e., $\sum_{M \in GL(n,\mathbb{Z})} p(M) = 1$):

Problem 3 (Average case version of Problem 2).

Given a matrix $M \in GL(n, \mathbb{Z})$ drawn with respect to the probability density p, efficiently recover M from MM^t (up to signed permutations of the columns) with high probability. (1.5)

In Section 2 we consider various methods of generating random bases of a lattice, which corresponds to different probability densities p, generalizing [1, §5.1.2] (see also Section 4). This is tantamount to looking for distributions for which Problem 3 is hard on average, much like SIS and LWE are average-case hard instances of variants of SVP and CVP, respectively. We then perform experiments on them in Section 3. Our main finding there is that a certain well-known existing method, namely generating matrices by multiplying unipotents (e.g., Magma's `RandomSLNZ` command), is cryptographically weak: we were able to recover M in instances in dimensions over 1000 that were in some measurable ways comparable to NTRU lattices having purported 256-bit quantum cryptographic strength. That gives an example of an average-case easy distribution. In the other direction, we also describe some other methods of generating random bases which are average-case stronger, as well as make some comments on strategies to generate a large random-looking matrix in $GL(n, \mathbb{Z})$ using a only moderate number of random bits ("entropy"). In Section 4 we present some evidence that the random basis generation method used in the DRS NIST Post-Quantum Cryptography submission [20] may have security weaknesses.

Acknowledgements: it is a pleasure to thank Huck Bennett, Leo Ducas, Nicholas Genise, Craig Gentry, Shai Halevi, Nadia Heninger, Jeff Hoffstein, Hendrik Lenstra, Amos Nevo, Phong Nguyen, Oded Regev, Ze'ev Rudnick, Alice Silverberg, Damien Stehlé, Noah Stephens-Davidowitz, and Berk Sunar for very helpful discussions. We are particularly indebted to Joe Silverman for kindly suggesting an earlier variant of Algorithm 4, which is very similar to the one we suggest here, and to Daniel J. Bernstein for important comments about the poor equidistribution provided by Algorithm 2. We are also grateful to Galen Collier of the Rutgers University Office of Advanced

Research Computing for his assistance, and to the Simons Foundation for providing Rutgers University with Magma licenses.

2 Choosing random elements of $GL(n,\mathbb{Z})$

We consider the problem of uniformly sampling matrices in a large box[2]

$$\Gamma_T \quad := \quad \{M=(m_{ij})\in GL(n,\mathbb{Z}) \ : \ |m_{ij}|\leq T\}\ , \quad T>0\,, \tag{2.1}$$

inside $GL(n,\mathbb{Z})$. For large T one has $\#\Gamma_T \sim c_n T^{n^2-n}$, for some positive constant c_n[3]. We now consider a series of algorithms to sample matrices in $GL(n,\mathbb{Z})$. The first, and most naive, does uniformly sample Γ_T but is prohibitively slow:

```
Algorithm 1.                                                        (2.2)
For each $1\leq i,j\leq n$ sample $m_{i,j}\in\mathbb{Z}\cap[-T,T]$ at random.
Let $M=(m_{ij})$.
Discard and repeat if $\det(M)\neq\pm1$, otherwise
return $M$.
```

Though we do not analyze it here, the determinant of such a randomly chosen matrix M is a very large integer, and highly improbable to be ±1 as required for membership in $GL(n,\mathbb{Z})$. One minor improvement that can be made is to first check that the elements of each row (and of each column, as well) do not share a common factor, which is a necessary condition to have determinant ±1. Nevertheless, this fails to seriously improve the extreme unlikelihood of randomly producing an integral matrix of determinant ±1.

Problem 4.

Find a nontrivial uniform sampling algorithm which substantially speeds up Algorithm 1. (2.3)

[2]One can consider other shapes, such as balls; boxes are convenient for our applications and for making more concise statements. The same problem for $SL(n,\mathbb{Z})$ is of course equivalent.

[3]See [11, Corollary 2.3] and [4, (1.14)] for more details on this surprisingly difficult result.

We note that some computer algebra packages already include commands for generating random elements of $GL(n,\mathbb{Z})$. In addition to its command `RandomSLnZ` which we shall shortly come to in Algorithm 2, Magma's documentation includes the command `RandomUnimodularMatrix` for fairly rapidly generating matrices in $GL(n,\mathbb{Z})$ (not $SL(n,\mathbb{Z})$ as the name indicates) having "most entries" inside a prescribed interval, but provides no further explanation. Even after accounting for a typo which switches the role of the command's arguments, we found in fact that most of the entries were *outside* the prescribed interval (the documentation's claims notwithstanding). Furthermore, the lattices constructed using this command appear to be much easier to attack than those generated by the closest analog considered here (Algorithm 4). SageMath's `random_matrix` command has a `unimodular` constructor (designed for teaching purposes) which does produce matrices in $GL(n,\mathbb{Z})$ whose entries are bounded by a given size, but it is not as fast as other alternatives and its outputs must satisfy further constraints. For these reasons we did not seriously consider `RandomUnimodularMatrix` and `random_matrix`.

Because Algorithm 1 is so slow, the rest of this section considers faster algorithms which do *not* uniformly sample Γ_T, some better than others.[4] For $1 \leq i \neq j \leq n$ let $E_{i,j}$ denote the elementary $n \times n$ matrix whose entries are all 0 aside from a 1 in the (i,j)-th position. Here as elsewhere the abbreviation i.i.d. stands for independently identically distributed.

Algorithm 2 (Random products of unipotents, such as Magma's `RandomSLNZ`).

Input: a size bound b and word length ℓ.
Return: a random product $\gamma_1 \cdots \gamma_\ell$, where each γ_j is chosen i.i.d. uniformly among all $n \times n$ matrices of the form $I_n + xE_{i,j}$, with $i \neq j$ and $x \in \mathbb{Z} \cap [-b, b]$. (2.4)

As we shall see, the matrices produced by Algorithm 2 have a very special form, creating a cryptographic weakness.

Algorithm 2 can be thought of as a counterpart to the LLL algorithm [15], which applies successive unipotent matrices and vector swaps to reduce lat-

[4]Unfortunately it is prohibitively complicated here to describe particular parameter choices matching the bound in (2.1).

tices. Although Algorithm 2 does not literally contain vector swaps, they are nevertheless present in the background because conjugates of γ_j by permutation matrices have the same form. In that light, the following algorithm can then be thought of as an analog of BKZ reduction [22], since it involves block matrices of size much smaller than n. Its statement involves embedding maps $\Phi_{k_1,\ldots,k_d} : GL(d,\mathbb{R}) \hookrightarrow GL(n,\mathbb{R})$ for size-d subsets $\{k_1,\ldots,k_d\} \subset \{1,\ldots,n\}$ given by the formula

$$(\Phi_{k_1,\ldots,k_d}(h))_{i'j'} \quad = \quad \begin{cases} h_{ij}, & \text{if } i' = k_i \text{ and } j' = k_j \text{ for some } i,j \leq d; \\ \delta_{i'=j'}, & \text{otherwise}, \end{cases} \tag{2.5}$$

where $h = (h_{ij}) \in GL(d,\mathbb{R})$.[5] The image of $\Phi_{k_1,\ldots,k_d}$ is a subgroup of $GL(n,\mathbb{R})$ isomorphic to $GL(d,\mathbb{R})$. (Of course we will only apply the map $\Phi_{k_1,\ldots,k_d}$ to elements of $GL(d,\mathbb{Z})$.)

Algorithm 3 (Random products of smaller matrices).

Input: a word length ℓ and fixed dimension $2 \leq d < n$ for which one can uniformly[a] sample $GL(d,\mathbb{Z})$ matrices in a large box.
Return: a random product $\gamma_1 \cdots \gamma_\ell$ in which each $\gamma_j \in GL(n,\mathbb{Z})$ is a matrix of the form $\gamma = \Phi_{k_1,\ldots,k_d}(\gamma^{(d)})$, where $\gamma^{(d)}$ is a uniformly sampled random element of $GL(d,\mathbb{Z})$ in a box as mentioned above, and $\{k_1,\ldots,k_d\}$ is a uniformly sampled random subset of $\{1,\ldots,n\}$ containing d elements. (2.6)

[a]More generally, one can consider non-uniform distributions as well.

The matrices $\gamma^{(d)}$ can also be described as random conjugates (by permutation matrices) of the matrices from $GL(d,\mathbb{Z})$, embedded into the upper-left corner of an $n \times n$ matrix.

We expect Algorithm 3 produces more-uniformly distributed matrices as d increases. The role of the parameter d is essentially to interpolate between Algorithms 1 and 2, though the $d = 2$ case of Algorithm 3 is not precisely identical to Algorithm 2: this is because $\gamma^{(2)}$ can be a much more general than simply a unipotent.

[5]The role of $GL(\cdot,\cdot)$ as opposed to $SL(\cdot,\cdot)$ here is again purely cosmetic.

Next we turn to the following method, which among the algorithms we considered seems the best at rapidly creating uniformly-distributed entries of matrices in $GL(n,\mathbb{Z})$. This algorithm was originally suggested to us by Joseph Silverman in a slightly different form, in which more coprimality conditions needed to be checked. It relies on the fact that an integral $n \times n$ matrix $M = (m_{ij})$ lies in $GL(n,\mathbb{Z})$ if and only if the n determinants of $(n-1)\times(n-1)$ minors

$$\det\begin{pmatrix} m_{22} & \cdots & m_{2n} \\ \vdots & \ddots & \vdots \\ m_{n2} & \cdots & m_{nn} \end{pmatrix},\ \det\begin{pmatrix} m_{21} & m_{23} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n3} & \cdots & m_{nn} \end{pmatrix},$$
$$\det\begin{pmatrix} m_{21} & m_{22} & m_{24} & \cdots & m_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & m_{n4} & \cdots & m_{nn} \end{pmatrix}, \ldots, \det\begin{pmatrix} m_{21} & \cdots & m_{2\,n-1} \\ \vdots & \ddots & \vdots \\ m_{n1} & \cdots & m_{n\,n-1} \end{pmatrix} \tag{2.7}$$

share no common factors.

Algorithm 4 (slight modification of a suggestion of Joseph Silverman).

Uniformly sample random integers $m_{i,j} \in [-T,T]$, for $2 \le i \le n$ and $1 \le j \le n$, until the n determinants in (2.7) share no common factor.
Use the euclidean algorithm to find integers $m_{11},\ldots,m_{1n}$ such that $\det((m_{ij})) = \pm 1$, the sign chosen uniformly at random.
Use least-squares to find the linear combination $\sum_{i\ge2}^{n} c_i[m_{i1}\cdots m_{in}]$ closest to $[m_{11}\cdots m_{1n}]$, and let $\widetilde{c_i}$ denote an integer nearest to c_i.
Return: the matrix M whose top row is

$$[m_{11}\cdots m_{1n}] - \sum_{i\ge2}^{n} \widetilde{c_i}[m_{i1}\cdots m_{in}]$$

and whose i-th row (for $i \ge 2$) is $[m_{i1}\cdots m_{in}]$. (2.8)

Remarks on Algorithm 4: The n integers in (2.7) are very large and unlikely to share a common factor between them: for example, the most probable common factor is 2, and this happens only with probability roughly 2^{-n}. Obviously the top row of M is chosen differently than the others, and its size is different as well since it is typically larger than size T – this is because the euclidean algorithm can produce large coefficients (as the minors in (2.7) are themselves so enormous). Also, it is likely that the first two or three minors will already be coprime, and hence that most of the entries in $[m_{11}\, m_{12}\, \cdots\, m_{1n}]$ will vanish. The use of rounding and least-squares cuts down this size and further randomizes the top row, while keeping the determinant equal to one.

One could instead try a different method to find an integral combination of the bottom $n-1$ rows closer to the initial guess for the top row. One extreme possibility involves appealing to the Closest Vector Problem (CVP) itself, which is thought to be very difficult. We found Algorithm 4 gave good randomness properties in that nearly all of the matrix is equidistributed, and it is fairly fast to execute. In comparison, we will see that using Algorithm 2 requires many matrix multiplications to achieve random entries of a similar size, which are not as well distributed.

The following algorithm is folklore (see [13, §4.1]). As we shall see just below, it shares some similarities with Algorithm 4.

Algorithm 5 (via Hermite Normal Form).

Create a uniformly distributed $m \times n$ matrix B, with $m \geq n$ and entries uniformly chosen in $\mathbb{Z} \cap [-T, T]$.
Decompose B in a Hermite normal form $B = UM$, where $M \in GL(n, \mathbb{Z})$ and $U = (u_{ij})$ has no nonzero entries with $i < j$.
Return: M. (2.9)

A surprising connection between Algorithms 4 and 5: Even though Algorithms 4 and 5 appear to be very different, they are actually extremely similar (in fact, arguably nearly identical) in practice. Algorithms for Hermite Normal Form (such as `HermiteDecomposition` in Mathematica) proceed by building the matrix M directly out of the rows of B whenever possible. For example, it is frequently the case that the first $n-1$ rows of U agree with those of the identity matrix I_n, or at least differ only very slightly; in other words, the first $n-1$ rows of B and M are expected to coincide or nearly coincide.[6] Also, the last row of U is an integral combination of the

[6]In our experiments, for example, the top $n-2$ rows agreed most of the time for $m = n \geq 10$.

first n rows of B. In contrast with Algorithm 4 this last combination, however, is mainly determined by arithmetic considerations, and in particular depends on the n-th row of B; thus more random information is used than in Algorithm 4, which uses only $n(n-1)$ random integers instead of the n^2 here.

To summarize, in fairly typical cases both Algorithms 4 and 5 populate the matrix M by generating all but one row uniformly at random, and use integral combinations to create a final row having relatively small entries. The practical distinction is essentially how this final row is created, which utilizes random information in Algorithm 5 but not in Algorithm 4. The final row also appears to be typically smaller (that is, closer to fitting in the box defined in (2.1)) when using Algorithm 4 than when using Algorithm 5; consequently, we did not do any experiments with Algorithm 5.

Note that the Hermite decomposition as stated above is not unique, since there are lower triangular matrices in $GL(n,\mathbb{Z})$. Thus there can be no immediate guarantee on the entry sizes of M unless this ambiguity is resolved. Algorithm 5 can be thought of as a p-adic analog of the following method of producing random rotations in $O(n)$: apply the Gram-Schmidt process to a matrix chosen according to a probability density function (e.g., Gaussian) which is invariant under multiplication by $O(n)$.

Remarks on an Algorithm in [21]**:** Igor Rivin makes the proposal in [21, §6.1] to generate matrices in $GL(n,\mathbb{Z})$ by applying complete lattice basis reduction to a basis of $\mathbb{R}^n$ chosen inside a large ball. Let $B \in GL(n,\mathbb{R})$ denote the $n \times n$ matrix whose rows consist of this basis. Complete lattice reduction produces a random element $\gamma \in GL(n,\mathbb{Z})$ of constrained size for which γB lies in a fixed fundamental domain for $GL(n,\mathbb{Z})\backslash GL(n,\mathbb{R})$.

This procedure is extremely slow, since complete lattice reduction is impractical in large dimensions. Rivin thus considers instead using weaker lattice basis reduction methods (such as LLL [15]) to speed this up, but at the cost of less-uniform distributions. For example, the results of LLL are thought to be skewed towards certain favored outputs avoiding "dark bases" [14]. Since our interest in generating random bases is to see how long incomplete lattice reduction takes on them, the use of complete lattice reduction to make the basis itself is too slow for our purposes (hence we did not consider this algorithm in our experiments).

3 Experiments on recognizing $\mathbb{Z}^n$

In this section we report on attempts to solve instances of Problem 2b which are generated using some of the algorithms from Section 2 for sampling $GL(n, \mathbb{Z})$. We first note that Geissler and Smart [8] reported on attempts using LLL [15] (as well as their own modification, for which they report up to a factor of four speedup), and concluded that LLL itself is insufficient for NTRU instances far smaller than those considered in (3.5) below. Here, however, we are considering various distributions of matrices on which LLL may be more successful (but yet still not solve). Instead of LLL alone, we try the following:

Procedure to test matrix generation algorithms with Problem 2b.

1. In Magma, apply LLL or Nguyen-Stehlé's L2 lattice basis reduction algorithm [19] to the Gram matrix $G = MM^t$, then
2. apply BKZ with incrementally-increasing block sizes $3, 4$, and 5.
3. Success is measured by whether or the output basis vectors all have norm equal to 1 (in which case they span a rotation of the $\mathbb{Z}^n$ lattice).

(3.1)

One can of course continue further with block sizes larger than 5, but we fixed this as a stopping point in order to be systematic.

Our main finding is that Algorithm 2 in Section 2 (as implemented in Magma's `RandomSLnZ`) is insecure for generating hard instances of Problem 2b. Algorithms 3, 4, and 5 fare much better.

It is not particularly surprising that LLL performs better on rotations of the $\mathbb{Z}^n$ lattice than it does on random lattices; for one thing, the random lattice in $\mathbb{R}^n$ has shortest vector on the order of $\sqrt{n}$, as opposed to 1 for rotations of the $\mathbb{Z}^n$ lattice (so the latter has an immense quantity of short vectors compared to typical lattices). Since LLL typically outperforms its provable guarantees, it is not surprising it is effective on Problem 2b. A detailed analysis of LLL and BKZ heuristics on Problem 2b is beyond the scope of this paper; rather, we emphasize they perform better on certain distributions in Problem 2b than on others.

A reference point for the bit-strength of lattice problems: NTRU

Before describing our experiments, a word is in order on how the security of lattice problems is measured. This is an active area in which no general consensus has been reached despite many competing suggestions (reflecting its underlying notoriously complicated difficulty). One area where bit strengths have been suggested is for the NTRU cryptosystem with particular parameters. We mention this to attempt to quantify the notion that lattice problems in high dimensions are hard and as a point of comparison, though there are of course many differences between NTRU lattices and rotations of the $\mathbb{Z}^n$ lattice (we don't say anything about the security of NTRU itself).

In more detail, NTRU matrices have the form

$$\begin{pmatrix} I_{n/2} & X \\ 0 & qI_{n/2} \end{pmatrix} \tag{3.2}$$

with n even, q an integer greater than one, and X chosen randomly from a certain distribution among all integral matrices of the form

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & \cdots & x_{n/2-1} & x_{n/2} \\ x_2 & x_3 & x_4 & \cdots & x_{n/2} & x_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n/2} & x_1 & x_2 & \cdots & x_{n/2-2} & x_{n/2-1} \end{pmatrix}, \ |x_j| \le \frac{q}{2}. \tag{3.3}$$

The rows of an NTRU matrix span an "NTRU lattice" $\Lambda \subset \mathbb{R}^n$. In [23] and in earlier NIST Post-Quantum Cryptography submissions the following quantum bit security is suggested for NTRU with the following parameters:

q	$n = \dim(\Lambda)$	estimated quantum security (in bits)
2048	886	128
2048	1486	256

(3.4)

These estimates are not directly relevant to the lattice bases we examine, which have different determinants and very different structure. Nevertheless, they are consistent with the general expectation that lattice problems in dimensions 500 or more (and especially 1,000 or more) become cryptographically difficult.

With the table in (3.4) in mind, we attempted to use the procedure (3.1) to solve Problem 2b on instances generated by Algorithms 2, 3, and 4 having similar size and density to NTRU matrices. Namely, in addition to having the same dimensions we also ensured (e.g., by taking sufficiently long products) that the vector lengths were at least those of the rows of the NTRU lattice basis in (3.2)[7], and generated using comparably many random bits ("entropy"). This overall amount of entropy involved in generating a random matrix is very important, since for long products such as in Algorithms 2 and 3 the entries of the resulting product matrix will be more uniformly distributed than if the random bits were packed into a smaller number of matrices, with each matrix using a large number of random bits. (For example, consider the extreme case where the product length is 1.)

3.1 Experiments with Algorithm 2 (Magma's `RandomSLnZ` command)

To mimic random elements of $GL(n,\mathbb{Z})$ and address the issue just mentioned, one may desire that the product matrix has as many nonzero entries as possible per random bit. For this reason, we set the parameter $b = 1$ in Algorithm 2 in order to take longer products (thereby further spreading out the nonzero entries of the matrix), while keeping the number of random bits constant. When the product length is less than the matrix size, one expects to have rows or columns of the product matrix which are unchanged by the successive matrix multiplications. (This much less likely to be the case for Gram matrices, however.)

Thus each random factor has at most a single nonzero off-diagonal entry, and this entry is in $\{-1, 0, 1\}$. It is prohibitive to pack in as many random bits as the total number of entries this way, since multiplication of large matrices is slow. As an extreme example, as part of a comparison with the last row of (3.4) we generated a random matrix in $GL(1486,\mathbb{Z})$ using products of length 55,000, again with $b = 1$. Generating the product alone took about half a day. Its row lengths were between 2^{14} and 2^{20} in size. For comparison, an NTRU matrix with similar parameters (as in Table 3.4) uses $\approx n/2 = 743$ random bits and roughly 1/4 of its entries are nonvanishing. One might hypothesize that having more random bits in the matrix makes

[7]The vector lengths of the rows in the NTRU matrix (3.2) are either roughly $\sqrt{\frac{n}{2}}\frac{q}{2}$ (for the first $n/2$ rows), or q (the last $n/2$ rows)

solving Problem 2b more difficult, but as we shall see this is in fact turns out not to be the case: the structure of the matrix plays a very important role, and the product structure from Algorithm 2 seems to be a weakness. That is, the larger the value of the parameter b, the more unusual properties the product matrix possesses.

We tried a number of experiments with Algorithm 2 from (2.4) using lattice reduction packages built into Magma. Our emphasis was mainly on the success of lattice reduction, but not its speed (these jobs ran from seconds to up to about a week, whether or not they were successful). As mentioned above, we took long products using Magma's `RandomSLnZ` command, each roughly matching the parameters of an NTRU instance whose hardness was estimated in (3.4). The following table shows cases where lattice reduction solved Problem 2b on instances generated by Algorithm 2 (using the testing procedure from (3.1), all with running time of a matter of days):

$n = \dim(\Lambda)$	estimated bit-hardness for corresp. NTRU (3.4)	range of vector lengths (in bits)	product (in bits)
886	128	from 25 to 32	55,000
1486	256	from 14 to 20	55,000

(3.5)

The second column reproduces the bit-strength estimates from (3.4) for NTRU lattices. Of course these are not the same lattices we are looking at here, so these numbers are given purely as a reference point. From the success of our trials one immediately sees that it is either much easier to solve Problem 2b than it is to crack NTRU, or that the matrices generated by Algorithm 2 give rise to fairly weak instances of Problem 2b. Either way, we conclude on the basis of this comparison that the Lenstra-Silverberg Problem 2b is fairly easy for matrices M generated by Magma's `RandomSLnZ` command.

3.2 Experiments with Algorithm 3 (random $SL(d,\mathbb{Z})$ matrices)

Next we consider matrices generated by Algorithm 3 (random $GL(d,\mathbb{Z})$'s), and find that for small d they are also cryptographically weak for the Lenstra-Silverberg problem, but stronger than those generated by Algorithm 2. Furthermore, we see they are stronger as the value of d increases.

The following tables list the outcome of several examples of experiments attacking instances of Problem 2b from matrices generated by Algorithm 3, where the matrices in $SL(d,\mathbb{Z}) \subset SL(n,\mathbb{Z})$ are chosen randomly among those whose matrix entries are bounded in absolute value by some parameter T, again following the procedure in (3.1). One sees the dramatic effect of the product length ℓ. For example, if ℓ is too short there may be rows and columns of the matrix not touched by the individual multiplications by the embedded random $d \times d$ matrices; if ℓ is too long, the matrix entries become large and lattice basis reduction becomes prohibitively slow (as so does generating the product to begin with).

n	d	T	ℓ	shortest row length (in bits)	longest row length (in bits)	found M?
200	2	1	4000	6.03607	12.7988	×
200	2	2	1500	1.29248	18.5329	✓
200	2	2	2000	7.86583	22.2151	×
200	2	3	1000	0.5	27.0875	×
200	2	3	2000	23.521	41.5678	×
200	2	10	500	2.04373	38.7179	✓
200	2	10	700	7.943	49.0346	×
200	3	1	1000	2.04373	11.3283	✓
200	3	1	1500	7.66619	17.1312	×
200	3	1	2000	13.0661	20.8768	×
200	3	2	500	3.27729	18.4087	✓
200	3	2	600	4.89232	24.111	×
200	3	2	1000	13.0585	34.0625	×
200	4	1	500	3.66096	12.2277	✓
200	4	2	300	0.5	24.2424	✓
200	4	2	400	1.79248	26.6452	×

key: n=lattice dimension, d=size of smaller embedded matrices, T=bound on embedded matrix entries, ℓ=length of the product of smaller matrices.

n	d	T	ℓ	shortest row length (in bits)	longest row length (in bits)	found M?
500	2	1	4000	0.	5.90085	✓
500	2	1	8000	3.41009	10.7467	✓
500	2	1	10000	7.08508	12.7447	✓
500	2	1	15000	12.6617	18.5326	✓
500	2	1	20000	18.0246	24.5732	×
500	2	2	4000	4.21731	18.587	✓
500	2	2	6000	12.3467	28.7882	×
500	2	2	8000	18.87	35.7267	×
500	2	2	10000	28.5508	45.8028	×
500	2	3	2000	0.	19.0752	✓
500	2	3	3000	7.38752	32.9895	✓
500	2	3	4000	16.9325	40.9656	×
500	2	10	1000	0.	30.3755	✓
500	2	10	2000	11.9964	61.5006	×
500	3	1	1000	0.	5.39761	✓
500	3	1	2000	1.29248	9.164	✓
500	3	1	3000	2.37744	13.9903	✓
500	3	1	4000	8.43829	17.4593	✓
500	3	1	5000	14.1789	21.528	✓
500	3	1	6000	18.3878	25.2578	×
500	3	1	7000	20.5646	29.287	×
500	3	2	1000	0.	15.551	✓
500	3	2	2000	3.24593	33.0945	✓
500	3	2	3000	23.5966	43.7986	×
500	3	3	1000	0.	28.1575	✓
500	3	3	2000	16.6455	53.1806	×
500	3	3	3000	41.3371	83.9486	×
500	4	1	1000	0.	9.85319	✓
500	4	1	2000	8.11356	18.9434	✓
500	4	1	3000	19.1019	26.9836	✓
500	4	1	4000	24.4869	35.6328	×
500	4	1	5000	26.6804	44.3982	×
500	4	1	6000	40.5944	53.3654	×
500	4	2	1000	6.29272	33.4373	✓
500	4	2	2000	33.6181	63.3469	×

n	d	T	ℓ	shortest row length (in bits)	longest row length (in bits)	found M?
886	2	1	3000	0	3.49434	✓
886	2	1	4000	0	3.80735	✓
886	2	1	5000	0	4.40207	✓
886	2	1	6000	0	5.30459	✓
886	2	1	7000	0	6.16923	✓
886	2	1	8000	0	6.90754	✓
886	2	1	9000	1	7.58371	✓
886	2	1	10000	2.37744	8.05954	✓
886	2	1	15000	5.46942	11.2176	✓
886	2	1	20000	8.6594	14.5837	✓
886	2	1	25000	10.884	18.035	✓
886	2	1	30000	15.0082	21.0333	✓
886	2	1	35000	17.6964	24.8408	✓
886	2	1	40000	20.7706	28.3888	✓
886	2	1	45000	24.484	30.6745	✓
886	2	1	50000	25.7401	34.0742	$\times$

Comments

Each sequence of experiments (for fixed values of n, d, and T) eventually fails when ℓ is sufficiently large. For ℓ too small the random product will not involve all the rows and columns of the matrix, meaning that the dimension of the lattice problem is effectively reduced to a smaller value of n. There is some correlation between entries having a short vector in M (the fifth column), especially in the trials having $n = 200$. For $n = 500$ one sees more successful trials with longer shortest rows, especially as d (and to a lesser extent, T) increase. Note that each entry in these tables corresponds to a single experiment; we did not attempt to average over several experiments since we want to report on the range of the row lengths (which is one means of comparison with NTRU lattices (3.2)).

We did not take values of $d > 4$, since it is difficult to use Algorithm 1 to generate larger random elements of $SL(n, \mathbb{Z})$.

The table for $n = 886$ is in some sense an elaboration of the middle entry of (3.5), the difference being that the latter uses unipotents (instead of embedded $SL(2, \mathbb{Z})$ matrices).

3.3 Experiments with Algorithm 4

Finally, we turn to the opposite extreme of random elements of $GL(n,\mathbb{Z})$ generated by Algorithm 4, in which the bottom $n-1$ rows are uniformly distributed among entries in the range $[-T,T]$. Here we were able to solve Problem 2b with instances having $n = 100$, even with entry sizes up to $T = 50$ (again, using the testing procedure in (3.1)). However, none of our experiments with $n \geq 110$ were successful at all, even with $T = 1$ (i.e., all entries below the top row are -1, 0, or 1).

n	T	shortest row length (in bits)	longest row length (in bits)	found M?
100	1	2.91645	4.65757	✓
100	3	4.14501	5.81034	✓
100	4	4.50141	6.20496	✓
100	10	5.64183	7.15018	✓
100	50	7.99332	9.77546	✓
100	1	2.91645	4.65757	✓
110	1	2.98864	4.54902	×
120	1	3.03304	4.77441	×
125	1	3.09491	4.93979	×
150	1	3.12396	5.09738	×
200	1	3.42899	5.32597	×
200	2	4.23584	6.42421	×
200	3	4.72766	6.82899	×
200	4	5.06529	7.41803	×

Comments

One numerically sees that matrices produced by Algorithm 2 are nearly rank-one matrices (i.e., up to a small overall perturbation relative to the size of the entries). In general, matrices in $GL(n,\mathbb{Z})$ with large entries have very small determinants (± 1) relative to their overall entry size, so they are already very close to singular matrices. However, the size of the rank of nearby matrices is important. The matrices produced by Algorithm 4 are instead perturbations of matrices having rank $n-1$ (which is as large as possible for singular $n \times n$ matrices). We expect Algorithm 3's matrices, which are produced by taking products of random $GL(d,\mathbb{Z})$ matrices, have intermediate behavior (but have not systematically analyzed this).

A related fact is that matrices produced by Algorithm 2 frequently have a very large row or column (if b is sufficiently large) – typically coming from the last factor in the matrix multiplication. That serves as a possible hint to recover the spelling of the word in the random product, along the lines of the length-based attack in [2, §4]. However, we were unable to turn this into a direct, general attack. For example, it is unclear what to do when the value of $x \in \mathbb{Z} \cap [-b, b]$ is small, say in the regime that $b \leq \ell$. (The situation is clearer when b is extremely large relative to ℓ, in which case we expect a bias effect in random words similar to underlying device used in [2, §4].)

4 Random basis generation in the DRS NIST Post-Quantum Cryptography competition submission

In [1, §5.1.2] some examples of methods for generating random lattice bases are described, which are closely related to Algorithms 2, 3, and 5. The authors reported their experiments on these methods resulted in similar outcomes in practice. Our experiments, which appear to possibly use slightly different variants of the algorithms used by the authors, do show a difference (as was explained in Section 3).

In this section we wish to make further comments about one method highlighted in [1], which is from the DRS NIST Post-Quantum competition submission [20, §2.2]. Random elements of $SL(n, \mathbb{Z})$ there are constructed as products of length $2R + 1$ of the form

$$M \quad = \quad P_1\gamma_1P_2\gamma_2P_3\gamma_3 \cdots P_R\gamma_RP_{R+1}\,, \tag{4.1}$$

where $P_1, \ldots, P_{R+1}$ are randomly chosen uniformly among permutation matrix in $SL(n, \mathbb{Z})$ and $\gamma_1, \ldots, \gamma_R$ are elements in $SL(n, \mathbb{Z})$ produced by the following random process. Let $A_+ = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ and $A_- = \left(\begin{smallmatrix} 1 & -1 \\ -1 & 2 \end{smallmatrix}\right)$. Then each γ_i is a block diagonal matrix with $\frac{n}{2}$ 2×2 entries chosen uniformly at random from $\{A_+, A_-\}$. This construction has some similarities with Algorithm 3 with $d = 2$, but is expected to be weaker in the sense that many of the $SL(2)$ matrices commute (being diagonal blocks of the same matrix). In fact, here there is essentially only one choice of diagonal entry (as A_+ is conjugate by $\left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ to A_-), and so one may take all the matrices on the block diagonal to be A_+ at the cost of allowing the P_i's to be signed permutation matrices). Alternatively, by rearranging the permutation matrices and applying an extra rotation on the right, Problem 2b on matrices of the form (4.1) is equivalent to it on products of the form

$$M' \;=\; M_1 M_2 \cdots M_R\,, \tag{4.2}$$

in which each M_i is conjugate of $\mathrm{diag}(\overbrace{A_+, A_+, \ldots, A_+}^{n/2})$ by a random signed permutation matrix.

Since Algorithm 3 with $d = 2$ performed relatively weakly in the experiments of Section 3, for these reasons we suspect Problem 2b is relatively easy to solve on matrices generated using (4.1) (as compared to those, say, generated using Algorithm 4). Our remaining comments in this section pertain solely to the construction of the element $M \in SL(n, \mathbb{Z})$ from (4.1) in the context of Problem 2b, and not to any other aspect of [20].

The parameters listed in [20, §3.2] assert 128-bit security for their scheme when $n = 912$ and $R = 24$. The testing procedure (3.1) easily solves Problem 2b when n or R are smaller yet still relatively large. For example, it took roughly an hour to recover M from MM^t when $(n, R) = (180, 24)$ using BKZ with block sizes up to 26.

Perhaps more interestingly, we also applied the testing procedure (3.1) for the parameters $n = 912$ and increasing values of R up to the recommended choice of $R = 24$ (see Figure 1). The results were strikingly successful, in that each trial for $R \leq 22$ successfully recovered M from MM^t using only LLL (without requiring BKZ). We additionally tried $R = 23$ and *nearly* recovered M using LLL this way: the longest vector in the LLL output had length $\sqrt{7}$, and running BKZ reduction with block size 3 for less than five minutes then found M. We were unsuccessful in the case $R = 24$ suggested in [20].

Again, these results are for Problem 2b applied to the random basis construction used in [20], not for the actual problem they consider there; nevertheless, this may indicate a weakness of the random bases in [20] as well. Interestingly, our experiments for fixed values of the product length R sometimes fared better for *larger* values of n. For example, we were not successful with $(n, R) = (200, 22)$ despite being successful for $(n, R) = (912, 22)$. Our explanation is that as n grows there may be a weakness in that it is hard to randomly fill out the full matrix M (analogous to the role of the length ℓ in Algorithms 2 and 3, where a similar issue was faced). Indeed, matrices

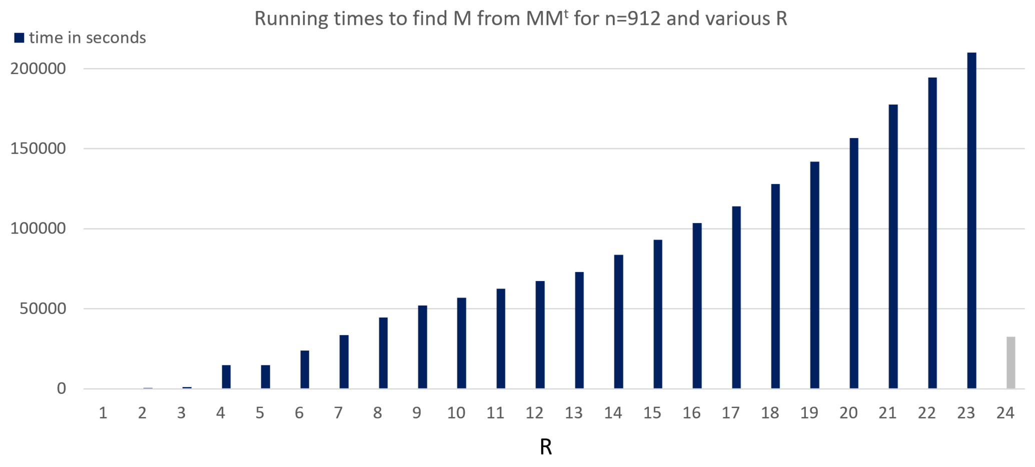

Figure 1: Suggested parameters in the DRS NIST submission [20] include $(n, R) = (912, 24)$. We experimentally tried to solve Problem 2b on instances generated by the random basis construction in [20, §2.2] with $n = 912$ and $1 \leq R \leq 24$. We were able to solve all cases for $R \leq 22$ in less than 60 hours using LLL, and the $R = 23$ case in slightly more time using the procedure in (3.1). Our attempts failed, though, for the recommended parameter $R = 24$. We conclude that DRS's method of random basis generation is insecure with the recommended parameter setting $(n, R) = (912, 24)$, at least for Problem 2b. Times are shown for runs on a Dell PowerEdge R740xd server equipped with two Intel Xeon Silver 4114 2.2GHz processors and 256GB RAM.

of the form (4.1) seem to have a very special form: Figure 2 shows the entry sizes in MM^t have a banded structure.

5 Conclusions

We have considered the role of generating random elements in $GL(n, \mathbb{Z})$ in the difficulty of lattice problems, and have found that it can have a profound influence. Concretely, Magma's `RandomSLnZ` command (Algorithm 2) is unsuitable in the context of the Lenstra-Silverberg's "Recognizing $\mathbb{Z}^n$ Decision" Problem 2b from (1.4). We were able to successfully attack lattices of dimension up to 1,486, which are in many ways comparable to NTRU lattices having claimed 256-bit quantum security. On the other hand, using appar-

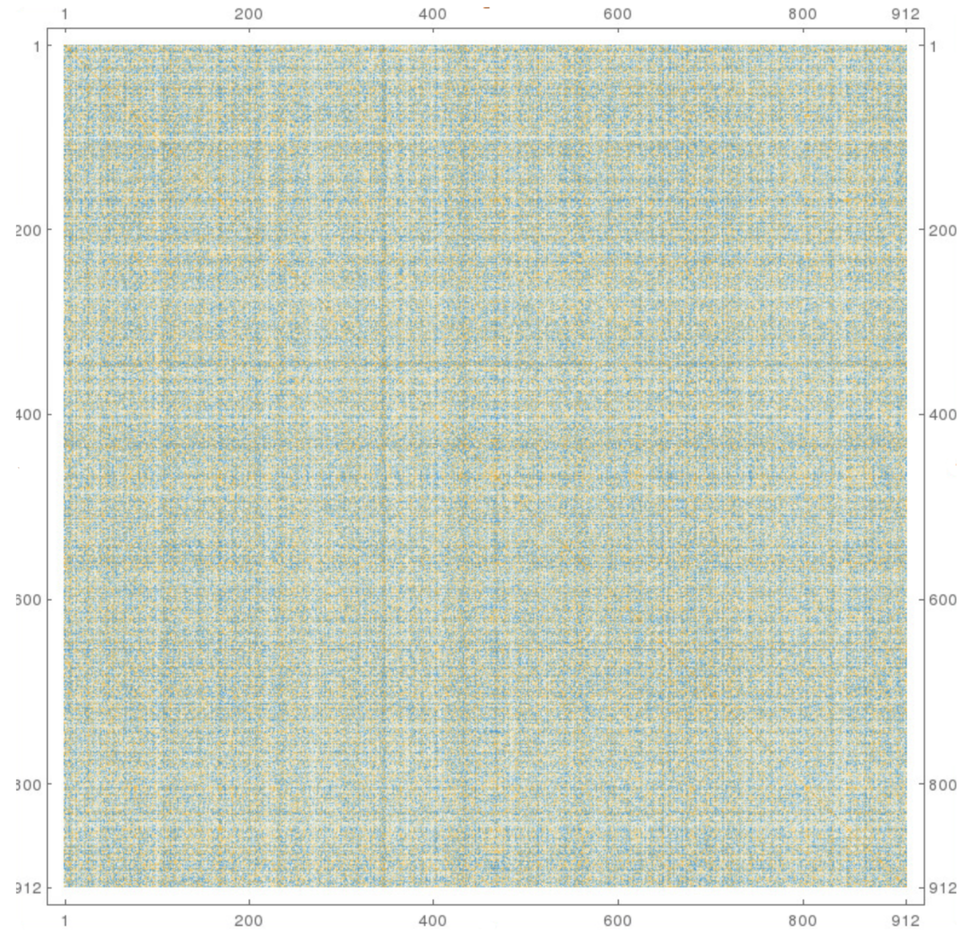

Figure 2: Mathematica's `MatrixPlot` command on a matrix the Gram matrix MM^t, where M is generated according to (4.1) with recommended parameters $n = 912$ and $R = 24$ from [20]. Similar plots arise when M is generated using Algorithm 3 with $d = 2$. In contrast, Gram matrices generated by Algorithm 4 have a (provably) far more uniform structure.

ently stronger methods such as Algorithms 3 and 4 make Problem 2b much more difficult to solve.

We would thus recommend not using Algorithm 2 in generating random bases for cryptographic applications. For similar reasons discussed in Section 4, we also recommend not using the random basis algorithm from the

NIST Post-Quantum Competition submission DRS [20] (e.g., we solved Problem 2 on instances of its random basis generation method using parameters just slightly below its recommend settings for 128-bit security).

We have not fully understood the weaknesses of these algorithms. It seems plausible that the failure to quickly fill out the matrix entries in a uniform way (many are zero at the initial stages) is at least partly to blame. For example, the construction of Algorithm 2 in some sense reverses the steps of an `LLL` basis reduction. Furthermore, one might expect the block sizes in Algorithm 3 to be related to the block sizes in the `BKZ` algorithm. From this point of view, we would expect Algorithms 4 and 5 to be the strongest lattice basis generation algorithms considered in this paper, consistent with our experiments.

References

[1] Yoshinori Aono, Thomas Espitau, and Phong Q. Nguyen, *Random Lattices: Theory And Practice*, preprint. `https://espitau.github.io/bin/random_lattice.pdf`

[2] Evgeni Begelfor, Stephen D. Miller, and Ramarathnam Venkatesan, *Non-abelian analogs of lattice rounding*, Groups Complexity Cryptology **7**, 117–133. Volume 7: Issue 2.

[3] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices, and Groups*, 3rd ed., Grundlehren der mathematischen Wissenschafter **290**, Springer, New York (1999).

[4] W. Duke, Z. Rudnick, and P. Sarnak, *Density of integer points on affine homogeneous varieties*, Duke Math. Jour. **71** (1993), 143–179.

[5] Noam D. Elkies, *A characterization of the $\mathbb{Z}^n$ lattice*, Math. Res. Lett. **2** (1995), 321–326.

[6] The FPLLL development team, *FPLLL, a lattice reduction library.* Available at `https://github.com/fplll/fplll`

[7] Steven Galbraith, *Mathematics of Public Key Cryptography*, `http://www.math.auckland.ac.nz/~sgal018/crypto-book/crypto-book.html`.

[8] Katharina Geißler and Nigel P. Smart, *Computing the $M = UU^t$ integer matrix decomposition*, Cryptography and Coding 2003, Lect. Notes in Comp. Sci. **2898**, Springer, Berlin Heidelberg, 2003, 223–233.

[9] C. Gentry and M. Szydlo, *Cryptanalysis of the revised NTRU signature scheme*, Advances in Cryptology—EUROCRYPT 2002, Lect. Notes in Comp. Sci. **2332**, Springer, Berlin, 2002, 299–320. `http://www.szydlo.com/ntru-revised-full02.pdf`

[10] Larry Gerstein, *Basic Quadratic Forms*, Graduate Studies in Mathematics **90**, Amer. Math. Soc.., Providence, RI, 2008.

[11] A. Gorodnik and A. Nevo, *The ergodic theory of lattice subgroups*, Annals of Mathematics Studies **172**, Princeton University Press, 2010.

[12] Christoph Hunkenschröder, *Deciding whether a Lattice has an Orthonormal Basis is in co-NP*, arxiv:1910.03838

[13] Seungki Kim, *On the shape of a high-dimensional random lattice*, Stanford Ph.D. Thesis (2015).

[14] Seungki Kim and Akshay Venkatesh, *The Behavior of Random Reduced Bases*, Int. Math. Res. Notices **2018**, pp. 6442–6480.

[15] Arjen K. Lenstra, Jr., Hendrik W. Lenstra, and Laszlo Lovasz, *Factoring polynomials with rational coefficients*, Mathematische Annalen, **261**, pp. 513–534, (1982).

[16] H. W. Lenstra Jr. and A. Silverberg, *Revisiting the Gentry-Szydlo Algorithm*, CRYPTO 2014, Lecture Notes in Computer Science, **8616**, Springer, Berlin, pp. 280–296.

[17] H. W. Lenstra Jr. and A. Silverberg, *Lattices with symmetry*, Journal of Cryptology **30** (2017), 760-804.

[18] H. W. Lenstra Jr. and A. Silverberg, *Testing isomorphism of lattices over CM-orders*, SIAM Journal on Computing **48**, no. 4 (2019), 1300–1334.

[19] Phong Q. Nguyen and Damien Stehlé, *An LLL algorithm with quadratic complexity*, SIAM J. Comput, **39**, pp. 874–903 (2009).

[20] Thomas Plantard, Arnaud Sipasseuth, Cédric Dumondelle, Willy Susilo, *DRS: Diagonal dominant Reduction for lattice-based Signature*, NIST Post-Quantum Digital Signature Competition entry, `https://csrc.nist.gov/Projects/post-quantum-cryptography/Round-1-Submissions`

[21] Igor Rivin, *How to pick a random integer matrix? (and other questions)*, Math. Comp. **85** (2016), 783–797.

[22] C.P. Schnorr, *A hierarchy of polynomial time lattice basis reduction algorithms*, Theoretical Computer Science **53** (1987), 201–224.

[23] William Whyte and Lee Wilson, *Quantum Safety In Certified Cryptographic Modules*, `https://icmconference.org/wp-content/uploads/A21c-Whyte.pdf`