Just how hard are rotations of $\mathbb{Z}^n$? Algorithms and cryptography with the simplest lattice

Huck Bennett[*] Atul Ganju[†] Pura Peetathawatchai[†] Noah Stephens-Davidowitz[†‡]

Abstract

We study the computational problem of finding a shortest non-zero vector in a rotation of $\mathbb{Z}^n$, which we call $\mathbb{Z}$SVP. It has been a long-standing open problem to determine if a polynomial-time algorithm for $\mathbb{Z}$SVP exists, and there is by now a beautiful line of work showing how to solve it efficiently in certain special cases. However, despite all of this work, the fastest known algorithm that is proven to solve $\mathbb{Z}$SVP is still simply the fastest known algorithm for solving SVP (i.e., the problem of finding shortest non-zero vectors in *arbitrary* lattices), which runs in $2^{n+o(n)}$ time.

We therefore set aside the (perhaps impossible) goal of finding an efficient algorithm for $\mathbb{Z}$SVP and instead ask what else we can say about the problem. E.g, can we find *any* non-trivial speedup over the best known SVP algorithm? And, what consequences would follow if $\mathbb{Z}$SVP actually is hard? Our results are as follows.

1. We show that $\mathbb{Z}$SVP is in a certain sense strictly easier than SVP on arbitrary lattices. In particular, we show how to reduce $\mathbb{Z}$SVP to an *approximate* version of SVP in the same dimension (in fact, even to approximate *unique* SVP, for any constant approximation factor). Such a reduction seems very unlikely to work for SVP itself, so we view this as a qualitative separation of $\mathbb{Z}$SVP from SVP. As a consequence of this reduction, we obtain a $2^{0.802n}$-time algorithm for $\mathbb{Z}$SVP, i.e., a non-trivial speedup over the best known algorithm for SVP on general lattices.
2. We show a simple public-key encryption scheme that is secure if (an appropriate variant of) $\mathbb{Z}$SVP is actually hard. Specifically, our scheme is secure if it is difficult to distinguish (in the worst case) a rotation of $\mathbb{Z}^n$ from *either* a lattice with all non-zero vectors longer than $\sqrt{n/\log n}$ *or* a lattice with smoothing parameter significantly smaller than the smoothing parameter of $\mathbb{Z}^n$. The latter result has an interesting qualitative connection with reverse Minkowski theorems, which in some sense say that "$\mathbb{Z}^n$ has the largest smoothing parameter."
3. We show a distribution of bases $\mathbf{B}$ for rotations of $\mathbb{Z}^n$ such that, if $\mathbb{Z}$SVP is hard for *any* input basis, then $\mathbb{Z}$SVP is hard on input $\mathbf{B}$. This gives a satisfying theoretical resolution to the problem of sampling hard bases for $\mathbb{Z}^n$, which was studied by Blanks and Miller [BM21]. This worst-case to average-case reduction is also crucially used in the analysis of our encryption scheme. (In recent independent work that appeared as a preprint before this work, Ducas and van Woerden showed essentially the same thing for general lattices [DvW21a], and they also used this to analyze the security of a public-key encryption scheme.)
4. We perform experiments to determine how practical basis reduction performs on different bases of $\mathbb{Z}^n$. These experiments complement and add to those performed by Blanks and Miller, as we work with a larger class of reduction algorithms (i.e., larger block sizes) and study the "provably hard" distribution of bases described above. We also observe a threshold phenomenon in which "basis reduction algorithms on $\mathbb{Z}^n$ nearly always find a shortest non-zero vector once they have found a vector with length less than $\sqrt{n}/2$," and we explore this further.

[*]Oregon State University. huck.bennett@oregonstate.edu. Part of this work was completed while the author was at the University of Michigan and supported by the National Science Foundation under Grant No. CCF-2006857. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the National Science Foundation.

[†]Cornell University.

[‡]noahsd@gmail.com. Supported in part by the National Science Foundation under Grant No. CCF-2122230.

Contents

1 Introduction

God made the integers; all the rest is the work of man.

Leopold Kronecker

A lattice $\mathcal{L} \subset \mathbb{R}^n$ is the set of all integer linear combinations of linearly independent basis vectors $\mathbf{B} := (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{R}^{n \times n}$, i.e.,

$$\mathcal{L} = \mathcal{L}(\mathbf{B}) = \{z_1 \boldsymbol{b}_1 + \cdots + z_n \boldsymbol{b}_n \ : \ z_i \in \mathbb{Z}\} \ .$$

Lattices have recently played a central role in cryptography, as many powerful cryptographic schemes have been constructed using lattices. (See [Pei16] and the references therein.) These schemes' security rests on the hardness of (worst-case) computational problems related to lattices, such as the Shortest Vector Problem (SVP), in which the goal is to find a non-zero lattice vector whose ℓ_2 norm is minimal, given a basis $\mathbf{B}$ for the lattice.

Perhaps the simplest example of a lattice is the *integer lattice* $\mathbb{Z}^n$, which has the identity matrix as a basis. Of course, the shortest non-zero vectors in $\mathbb{Z}^n$ are simply the standard basis vectors and their negations $\pm \boldsymbol{e}_1, \ldots, \pm \boldsymbol{e}_n$, which have length one. So, it is trivially easy to find a shortest non-zero vector in $\mathbb{Z}^n$, and other computational lattice problems are also easy when the relevant lattice is $\mathbb{Z}^n$.

However, suppose that we are given some basis $\mathbf{B}$ for a *rotation* of $\mathbb{Z}^n$, i.e., a basis $\mathbf{B}$ such that the lattice $\mathcal{L}(\mathbf{B})$ generated by this basis is $R\mathbb{Z}^n$ for some orthogonal matrix $R \in \mathsf{O}_n(\mathbb{R})$. Of course, if the basis $\mathbf{B}$ is simply R itself, then it is still easy to find a short vector in $\mathbb{Z}^n$. (E.g., any column of R will do.) But, it does not need to be so easy. For example, the lovely matrix

$$\mathbf{B} := \begin{pmatrix} 3\sqrt{3898} & -5382\sqrt{\frac{2}{1949}} & \frac{31195}{\sqrt{3898}} & \frac{15857}{3} \cdot \sqrt{\frac{2}{1949}} \\ 0 & \sqrt{\frac{682378}{1949}} & -110727\sqrt{\frac{2}{664977361}} & \frac{676011}{\sqrt{1329954722}} \\ 0 & 0 & \sqrt{\frac{64221}{682378}} & \frac{67240}{3} \cdot \sqrt{\frac{2}{21911498769}} \\ 0 & 0 & 0 & \frac{1}{3\sqrt{128442}} \end{pmatrix}$$

is a basis for a rotation of $\mathbb{Z}^4$, but it is not immediately clear how to find a vector of length one in the lattice generated by $\mathbf{B}$.[1] We write $\mathbb{Z}$SVP for the problem of finding vectors of length one in a rotation $\mathcal{L}$ of $\mathbb{Z}^n$, given a basis for $\mathcal{L}$.

Indeed, this is a well known problem, and it has been a long-standing open problem to settle the complexity of $\mathbb{Z}$SVP, leading to a beautiful line of work [GS02, Szy03, GS03, LS14, LS17, CGG17, Hun19]. Frustratingly, despite all of this wonderful work, the fastest known algorithm that is proven to solve $\mathbb{Z}$SVP is still simply the fastest known algorithm that is proven to solve SVP on arbitrary lattices, a $2^{n+o(n)}$-time algorithm [ADRS15]. So, we do not even know whether $\mathbb{Z}$SVP is any easier than SVP on arbitrary lattices, let alone whether there exists, e.g., a polynomial-time algorithm!

1.1 Our results

In this paper, we set aside the question of whether a polynomial-time algorithm for $\mathbb{Z}$SVP exists and instead ask what else we can say about $\mathbb{Z}$SVP. Specifically, we study the following questions.

1. Can we at least solve $\mathbb{Z}$SVP in time better than $2^{n+o(n)}$? (In other words, can we at least do better than just plugging in an algorithm that solves SVP on all lattices?)

2. If it is hard to solve $\mathbb{Z}$SVP (or variants of it), does this imply any interesting cryptography?

[1] Of course, this is not actually a hard problem for a computer, since it is only four-dimensional and SVP can be solved efficiently when the dimension n is constant. Indeed, one example of a unit length vector in this lattice is $\mathbf{B}\boldsymbol{z}$, where $\boldsymbol{z} := (59, 396, 225, -326)^T$.

3. In particular, is there some (efficiently sampleable) distribution of instances of $\mathbb{Z}$SVP such that these instances are provably hard if $\mathbb{Z}$SVP is hard in the worst case? I.e., is there a "hardest possible" distribution of bases?

4. Do known algorithms perform any differently on rotations of $\mathbb{Z}^n$ empirically?

We essentially give positive answers to all of these questions, giving a richer perspective on $\mathbb{Z}$SVP and related problems, as we detail below.

Provably faster algorithms for $\mathbb{Z}^n$. Our first main result, presented in Section 5, is an exponential-time algorithm for $\mathbb{Z}$SVP that is faster than the fastest known algorithm for SVP over arbitrary lattices. In fact, we show something significantly stronger: an efficient dimension-preserving reduction from $\mathbb{Z}$SVP to γ-approximate SVP over general lattices for any constant $\gamma = O(1)$. In other words, we show that in order to find an exact shortest vector in a rotation of $\mathbb{Z}^n$, it suffices to find an *approximate* shortest vector in an arbitrary lattice. (In fact, we reduce to the γ-*unique* Shortest Vector Problem, which is SVP in which the shortest vector is guaranteed to be a factor of γ shorter than "the second shortest vector," appropriately defined.)

Theorem 1.1 (Informal. See Corollary 5.3)**.** *There is an efficient reduction from $\mathbb{Z}$SVP to γ-approximate SVP (in fact, to γ-unique SVP, a potentially easier problem) in the same dimension for any constant $\gamma = O(1)$.*

If we plug in the fastest known algorithm for $O(1)$-approximate SVP, we immediately obtain a $2^{0.802n}$-time provably correct algorithm for $\mathbb{Z}$SVP [LWXZ11, WLW15, AUV19]. (And, under a purely geometric conjecture, we obtain a running time of $(4/3)^{n+o(n)} \approx 2^{0.415n}$ [Ste20].)

However, the specific running times are perhaps less interesting than the high-level message: solving exact SVP over rotations of $\mathbb{Z}^n$ is no harder than solving *approximate* (or even *unique*) SVP over arbitrary lattices in the same dimension. We certainly do not expect such a reduction to work for arbitrary lattices, so this proves that there is something inherently "easier" about $\mathbb{Z}^n$.

In fact, there is nothing particularly special about polynomial-time reductions in this context, and we more generally achieve a smooth trade-off between the running time of the reduction and the approximation factor γ in the resulting SVP instance. In particular, we can reduce $\mathbb{Z}$SVP to γ-SVP in time roughly $(n/\gamma^2)^{\gamma^2}$ for any $\gamma \leq \sqrt{n}/2$ (using roughly $(n/\gamma^2)^{\gamma^2}$ queries to a γ-SVP oracle).

Indeed, our reduction solves SVP over any lattice $\mathcal{L}$ that has "remarkably few approximately shortest points." The running time depends on exactly how many γ-approximate shortest vectors $\mathcal{L}$ has. For example, this yields essentially the same trade-off for any (rotation of a) lattice that is the direct sum of many low-dimensional lattices, and any small perturbation of $\mathbb{Z}^n$. The key tool that we use here is lattice sparsification, which was originally developed by Khot (in a rather different context) [Kho05].

We also present (in Section 5.1) a simple reduction from $\mathbb{Z}$SVP to $\sqrt{2}$-SVP. (Notice that a reduction from $\mathbb{Z}$SVP to γ-SVP for $\gamma < \sqrt{2}$ is trivial, but a reduction for $\gamma \geq \sqrt{2}$ is non-trivial.) This reduction is formally weaker than the one described in Theorem 1.1 (since it only works for the approximation factor $\sqrt{2}$), but it is simpler and more intuitive (and it also has the benefit of being deterministic). We hope that future authors might generalize it to work for larger approximation factors, perhaps even superconstant approximation factors. (In fact, we know how to extend it to the approximation factors $\sqrt{3}$ and $2 = \sqrt{4}$, but our proof relies on tedious case analysis, so we do not bother to include this result.)

Our reduction can also potentially be viewed as a sort of hardness proof for unique SVP (uSVP), which is a key problem in lattice cryptography. Despite its importance, little is known about its hardness under standard complexity-theoretic assumptions: uSVP is not even known to be NP-hard for any constant approximation factor greater than 1 [AD16, Ste16b]. However, our reduction shows that uSVP is hard for *any* constant approximation factor if $\mathbb{Z}$SVP is hard. (We show essentially the same result for the Bounded Distance Decoding problem (BDD), which is closely related to uSVP via known reductions [LM09, BSW16].) We emphasize that hardness of $\mathbb{Z}$SVP is a non-standard and perhaps even overly strong assumption, and so, while notable as the first of its kind, this result by itself should be viewed as relatively weak evidence that approximate uSVP is hard.

A public-key encryption scheme. Our next main result, presented in Section 4, is a public-key encryption scheme whose security can be based on the (worst-case) hardness of variants of $\mathbb{Z}$SVP. To be clear, we do *not* recommend using this scheme in practice, as its security rests on the hardness of problems that might very well turn out to be easy!

Specifically, we show an encryption scheme that is secure if it is difficult to distinguish a rotation $\mathbb{Z}^n$ *either* from (1) a lattice with no non-zero vectors with length less than roughly γ for $\gamma \approx \sqrt{n/\log n}$; or (2) from a lattice with smoothing parameter $\eta_\varepsilon(\mathcal{L})$ smaller than $\eta_\varepsilon(\mathbb{Z}^n)/\alpha$ for any $\alpha > \omega(1)$. (See Section 2.2 for the definition of the smoothing parameter.) We call these problems γ-$\mathbb{Z}$GapSVP and α-$\mathbb{Z}$GapSPP, respectively.

Theorem 1.2 (Informal, see Theorem 4.8)**.** *There is a public-key encryption scheme that is secure if* either *γ-$\mathbb{Z}$GapSVP or α-$\mathbb{Z}$GapSPP is hard, for $\gamma \approx \sqrt{n/\log n}$ and any $\alpha > \omega(1)$.*

We stress that both $\mathbb{Z}$GapSVP and $\mathbb{Z}$GapSPP are *worst-case* problems. In particular, our encryption scheme is secure unless there is a polynomial-time algorithm that distinguishes *all* bases of rotations of $\mathbb{Z}^n$ from *all* lattices that either have no short vectors or have small smoothing parameter. (A critical step in our proof is a worst-case to average-case reduction showing how to sample a basis for a rotation of $\mathbb{Z}^n$ that is provably as secure as *any* basis. We discuss this more below.)

We note that the approximation factor $\gamma \approx \sqrt{n/\log n}$ might look quite impressive at first. Specifically, prior work shows public-key encryption schemes that are secure if γ'-GapSVP (as opposed to γ-$\mathbb{Z}$GapSVP) is hard for $\gamma' \approx n^{3/2}$, where γ'-GapSVP asks us to distinguish a lattice with a non-zero vector with length at most one from a lattice with no non-zero vectors with length less than γ'. So, our approximation factor $\gamma \approx \sqrt{n/\log n}$ seems much better. (And, perhaps it is. In particular, we do not know algorithms that solve γ-$\mathbb{Z}$GapSVP faster than γ'-GapSVP or even γ-GapSVP.)

Of course, our reduction only works for γ-$\mathbb{Z}$GapSVP, which is potentially a much easier problem that γ-GapSVP, or even than γ'-GapSVP. (Indeed, we are certainly not even willing to conjecture that $\mathbb{Z}$SVP is hard, let alone γ-$\mathbb{Z}$GapSVP.) And, from another perspective, the approximation factor of $\gamma \approx \sqrt{n/\log n}$ seems rather weak. Specifically, since $\mathbb{Z}^n$ (and any rotation of $\mathbb{Z}^n$) has determinant one, it is trivial by Minkowski's theorem to distinguish a rotation of $\mathbb{Z}^n$ from a lattice with no non-zero vectors with length less than roughly $\sqrt{n}$. So, from this point of view, our approximation factor γ is just a factor of $\sqrt{\log n}$ from trivial.

The approximation factor α for $\mathbb{Z}$GapSPP is harder to interpret, in part because computing the smoothing parameter is not nearly as well studied as computing the length of the shortest non-zero vector. (But, see [CDLP13].) However, a recent series of works [DR16, RS17, ERS21, RS21] has shown that there is a certain sense in which "$\mathbb{Z}^n$ has the largest smoothing parameter of any lattice." (E.g., up to a pesky factor of 2, $\mathbb{Z}^n$ is known to have the largest smoothing parameter of any determinant-one lattice whose gram matrix is integral [RS21].) So, there is a certain vague sense in which α-$\mathbb{Z}$GapSPP is the problem of "recognizing $\mathbb{Z}^n$ by one of its most distinguishing features," and we therefore think of it as an approximate analogue of the problem of simply recognizing a rotation of $\mathbb{Z}^n$ (i.e., distinguishing a rotation of $\mathbb{Z}^n$ from any other lattice). See Section 4.4.

In fact, we note in passing that our cryptographic scheme can be adapted to work with other lattices, but it seems that $\mathbb{Z}^n$ offers distinct advantages here (setting aside the question of whether the scheme is actually secure).

Sampling provably secure bases. Our next main result, presented in Section 3, is a way to sample a "hardest possible" basis $\mathbf{B}$ for a rotation of $\mathbb{Z}^n$. For example, we show an explicit (efficiently sampleable) distribution of bases $\mathbf{B}$ for rotations of $\mathbb{Z}^n$ such that, if it is hard to solve $\mathbb{Z}$SVP in the worst case, then it is hard to solve $\mathbb{Z}$SVP on input $\mathbf{B}$. The basic idea is to use the discrete Gaussian sampling algorithm of [GPV08] to use any basis of a rotation $\mathcal{L}$ of $\mathbb{Z}^n$ to obtain many discrete Gaussian samples from $\mathcal{L}$, sufficiently many that we have a generating set of $\mathcal{L}$. We can then apply any suitable algorithm that converts a generating set into a basis. (A similar idea was used in [HR14] in a different context. More recently, in independent work that was published on eprint before this work, [DvW21a] used more-or-less the same idea in a very similar context. See Section 1.2.)

This gives a theoretically rigorous answer to the question studied by Blanks and Miller [BM21], who considered the relative hardness of solving $\mathbb{Z}$SVP for different input bases. We show that there is a relatively simple input distribution that is provably as hard as any other.

Indeed, we have already implicitly mentioned this result, as it is crucially used in the security reductions for our encryption scheme.

Experimental results for $\mathbb{Z}$SVP Our final contribution, presented in Section 6, consists of a number of experimental results showing how practical heuristic lattice algorithms perform on $\mathbb{Z}^n$.

Our first such set of experiments ran state-of-the-art basis reduction algorithms on bases of $\mathbb{Z}^n$ that were generated in different ways and compared their effectiveness.[2] These experiments complement similar experiments performed by Blanks and Miller [BM21]. Our experiments differ from those of Blanks and Miller in that we used the BKZ algorithm with larger block sizes; performed more trials; and performed experiments on the distribution of bases resulting from our worst-case to average-case reduction.

Here, our results were broadly comparable to those of [BM21]. See Section 6.1 for the details. However, we note that our new experiments on the distribution of bases resulting from worst-case to average-case reductions suggest that these bases achieve comparable security to the bases studied in [BM21] with *much* shorter vectors.

In conducting these experiments, we noticed a curious *threshold phenomenon* exhibited by basis reduction algorithms when run on $\mathbb{Z}^n$. Specifically, we noticed that the output of these algorithms nearly always either contained a vector of length one or contained no vectors with length less than roughly $\sqrt{n}/2$. This suggests that, once a basis reduction finds a vector in $\mathbb{Z}^n$ with length significantly less than $\sqrt{n}/2$, it nearly always finds a shortest vector.

Our second set of experiments therefore studies this phenomenon specifically. Indeed, we show that the behavior is quite striking. See Section 6.2. We offer some rough intuition for why this might happen when one performs basis reduction on bases of $\mathbb{Z}^n$, and [DvW21a] predict essentially the same phenomenon more generally (for any "unusual" lattice).

1.2 Related work

As we mentioned above, there is by now a beautiful sequence of works showing polynomial-time algorithms for certain special cases of $\mathbb{Z}$SVP [GS02, GS03, LS14, LS17, CGG17]. A summary of their results is beyond the scope of this work, but we note that their techniques are very different from those in this work with the exception of Szydlo's heuristic algorithm [Szy03]. In particular, Szydlo presented a heuristic algorithm that solves $\mathbb{Z}$SVP by finding many vectors of length roughly $c\sqrt{n}$ (where the constant $c > 0$ is unspecified), which can be viewed as a heuristic reduction from $\mathbb{Z}$SVP to $c\sqrt{n}$-SVP. In contrast, we give an efficient reduction with a proof of correctness from $\mathbb{Z}$SVP to γ-uSVP for any constant γ (and, more generally, a roughly $(n/\gamma^2)^{\gamma^2}$-time reduction for $\gamma \le \sqrt{n}/2$).

Our public-key encryption scheme is quite similar to a scheme recently proposed by Ducas and van Woerden [DvW21a], in a beautiful independent work that appeared as a preprint before the present work was finished. On one hand, Ducas and van Woerden's construction is more general than ours—it works with any "remarkable" lattice, of which $\mathbb{Z}^n$ is an example. (We do note in passing that our constructions make sense for a more general class of lattices, but we do not attempt to make this precise.) On the other hand, because we specialize to $\mathbb{Z}^n$, our scheme is arguably simpler, and the hardness assumptions that we require for security, while formally incomparable, are arguably weaker.

Perhaps the biggest difference is that in [DvW21a], the ciphertext is a target point that is very close to the lattice, effectively within the unique decoding radius of $\mathbb{Z}^n$, i.e., $1/2$ (or for more general lattices, within whatever radius one can efficiently decode, uniquely). And, the [DvW21a] decryption algorithm recovers the unique lattice vector within this distance of the target point. In this context, $\mathbb{Z}^n$ is not a particularly good lattice because its unique decoding radius is rather small (relative to, e.g., its determinant). (Of course, Ducas

[2]Note that we ran these experiments directly on bases of $\mathbb{Z}^n$, rather than on rotations of bases of $\mathbb{Z}^n$ because the algorithms themselves are rotation invariant.

and van Woerden list many "remarkable" lattices.) In contrast, our ciphertext is a target point that is quite far away from the lattice, at distance $\Theta(\sqrt{n})$ (well above the radius at which unique decoding is possible), and our decryption algorithm simply determines whether the target is closer or farther than a certain threshold value. Because of this difference, our scheme achieves security under arguably weaker hardness assumptions (because we work at much larger radii), but each of our ciphertexts encodes just a single-bit plaintext, while [DvW21a] encode many plaintext bits in each ciphertext (or, more accurately, they construct a KEM). The assumptions are not directly comparable, however, as [DvW21a]'s hardness assumptions concern the lattice $\mathbb{Z}^n \oplus \alpha\mathbb{Z}^n$ for a cleverly chosen scaling factor α, whereas our hardness assumptions work with $\mathbb{Z}^n$ directly. Ducas and van Woerden also show a signature scheme and a zero-knowledge proof, while we do not.

Ducas and van Woerden's work also contains more-or-less the same worst-case to average-case reduction that we describe in Section 3, and therefore also more-or-less the same distribution of bases that we propose. Indeed, in this case their work is essentially strictly more general than ours. (Similar ideas also appeared in [HR14], though in a different context. Our proofs in Section 3 immediately generalize to other lattices.)

Blanks and Miller introduced two of the basis-generating procedures that we study, and performed experiments on them to determine if basis reduction algorithms could break them [BM21]. Our empirical work on different bases for $\mathbb{Z}^n$ is best viewed as follow-up work to [BM21]. In particular, we perform more trials and run BKZ with larger block sizes. Additionally, we perform experiments on the discrete Gaussian bases described above, which were not considered in [BM21].

1.3 A brief note on using rotated bases as opposed to, e.g., Gram matrices

Throughout this paper, we work with bases $\mathbf{B}$ that are *rotations* of bases of $\mathbb{Z}^n$, or more precisely, orthogonal transformations of bases of $\mathbb{Z}^n$. And, we sometimes even work with uniformly random orthogonal transformations. We do this largely because it is convenient for our presentation and proofs. Of course, to be fully formal, we must specify exactly the input format that we use for these bases, which in general will not be rational. Indeed, a true sample from the uniform distribution over orthogonal transformations will not even admit a finite description. We adopt the convention, common in the literature on lattices, of ignoring these issues. They can be resolved by appropriately discretizing the space.

There are at least two alternative approaches, which admittedly have some major advantages. In particular, neither approach runs into the representation issues described above.

One alternative approach is to work with the *Gram matrix* $\boldsymbol{G} := \mathbf{B}^T\mathbf{B}$ instead of the basis $\mathbf{B}$ itself. Notice that the Gram matrix is rotation independent—i.e., if $R \in \mathsf{O}_n(\mathbb{R})$ is an orthogonal matrix, then $\mathbf{B}^T\mathbf{B} = (R\mathbf{B})^T R\mathbf{B}$, so that working with the Gram matrix effectively removes the need to discuss rotations. And, with some care, one can move freely between Gram matrices and bases. The Gram matrix is also easy to represent in bits, because the Gram matrix is always an integer matrix when $\mathbf{B}$ generates a rotation of $\mathbb{Z}^n$. In fact, we do explicitly work with the Gram matrix when we present our cryptographic scheme, since in that case we are actually proposing an explicit construction. However, most of the literature (and most of the results from prior work that we rely on) is written in terms of bases, not Gram matrices. So, we (mostly) stick to working with bases.

Another approach is to work with some *canonical* rotation. For example, for any basis $\mathbf{B}$, there is a unique upper-triangular matrix $\mathbf{B}'$ with positive entries along the diagonal such that $\mathbf{B}' = R\mathbf{B}$ for some orthogonal transformation $R \in \mathsf{O}_n(\mathbb{R})$. This is the QR-decomposition of the basis (where, confusingly, Q in the QR-decomposition is an orthogonal transformation and R is an upper-triangular matrix with positive entries along the diagonal) or equivalently as the Cholesky decomposition of the Gram matrix. So, rather than work with arbitrary rotations, we could work with these canonical rotations. Indeed, this has a lot of appeal because many lattice algorithms (e.g., LLL) compute the QR-decomposition anyway, and though it does not consist of integers, it does consist of square roots of rational numbers. This would likely be our preferred approach if it did not require extra background knowledge for the reader.

Acknowledgements

The authors would like to thank Léo Ducas, Stephen D. Miller, and Michael Walter for helpful comments. In particular, we thank Léo Ducas for bringing our attention to the question of whether there exist lattices $\mathcal{L}$ in the same genus as $\mathbb{Z}^n$ with large $\lambda_1(\mathcal{L})$ (as discussed in Section 4.5). We also thank Phong Q. Nguyen for sharing an early version of [NP21] with us.

2 Preliminaries

We write I_n for the identity matrix. We write $\mathsf{O}_n(\mathbb{R})$ for the set of all orthogonal linear transformations. That is $\mathsf{O}_n(\mathbb{R})$ is the set of matrices $R \in \mathbb{R}^{n\times n}$ with the property that $R^T R = I_n$. We often informally refer to orthogonal transformations as "rotations." We refer to integer-valued matrices with determinant ± 1 (i.e, matrices in $\mathrm{GL}_n(\mathbb{Z})$) as *unimodular*.

2.1 Basic lattice definitions

We say that a lattice $\mathcal{L} = \mathcal{L}(\mathbf{B}) \subset \mathbb{R}^n$ with basis $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{R}^{n\times n}$ has *dimension* n. We use $\lambda_1(\mathcal{L})$ to denote the *minimum distance* of $\mathcal{L}$ (equivalently, the length of the shortest non-zero vector in $\mathcal{L}$). I.e., $\lambda_1(\mathcal{L}) := \min_{\boldsymbol{x}\in\mathcal{L}\setminus\{\mathbf{0}\}} \|\boldsymbol{x}\|$. More generally, we define the *ith successive minimum* $\lambda_i(\mathcal{L})$ for $1 \le i \le n$ (where n is the dimension of the lattice) to be the smallest value of $r > 0$ such that $\mathcal{L}$ contains at least i linearly independent vectors of length r:

$$\lambda_i(\mathcal{L}) := \min\{r > 0 : \dim(\mathrm{span}(\mathcal{L} \cap r\mathcal{B}_2^n)) \ge i\} \ .$$

Here $\mathcal{B}_2^n$ denotes the Euclidean unit ball in n dimensions.

Given a lattice $\mathcal{L}$ with basis $\mathbf{B}$, we define the *Gram matrix* of $\mathbf{B}$ to be $G := \mathbf{B}^T\mathbf{B}$. We define the *determinant* of such a lattice $\mathcal{L}$ to be $\det(\mathcal{L}) := |\det(\mathbf{B})| = \sqrt{\det(G)}$. We note that $\det(\mathcal{L})$ is well-defined because all bases of $\mathcal{L}$ are equivalent up to multiplication by unimodular matrices. Minkowski's Theorem upper bounds the minimum distance of a lattice in terms of its determinant, thereby relating the two most important lattice invariants. In particular, if $\mathcal{L}$ is a lattice of dimension n, Minkowski's Theorem asserts that

$$\lambda_1(\mathcal{L}) \le C\sqrt{n} \cdot \det(\mathcal{L})^{1/n} \qquad (1)$$

for some explicit constant $C > 0$.

2.2 The continuous and discrete Gaussian distributions and the smoothing parameter

For a vector $\boldsymbol{y} \in \mathbb{R}^n$ and parameter $s > 0$, we write

$$\rho_s(\boldsymbol{y}) := \exp(-\pi\|\boldsymbol{y}\|^2/s^2)$$

for the Gaussian mass of $\boldsymbol{y}$ with parameter s. We write D_s^n for the symmetric continuous Gaussian distribution on $\mathbb{R}^n$, that is, the distribution with probability density function given by

$$\Pr_{\boldsymbol{X}\sim D_s^n}[\boldsymbol{X} \in S] = \frac{1}{s^n} \cdot \int_S \rho_s(\boldsymbol{y})\mathrm{d}\boldsymbol{y} \ .$$

We simply write D_s for D_s^1.

We prove the following lemma in Appendix A.

Lemma 2.1. *For any $s > 0$, positive integer n, and $\varepsilon > \varepsilon_0$*

$$\Pr_{\boldsymbol{X}\sim D_s^n}[|\mathrm{dist}(\boldsymbol{X}, \mathbb{Z}^n)^2 - \nu| > \varepsilon n] \le 2\exp(-(\varepsilon - \varepsilon_0)^2 n/10) \ ,$$

where

$$\nu := \frac{n}{12} - \frac{\exp(-\pi s^2)}{2\pi^2} \cdot n \ ,$$

and

$$\varepsilon_0 := \frac{\exp(-4\pi s^2)}{6} \cdot (1 + 1/s^2) \ .$$

The Gaussian mass of a lattice $\mathcal{L} \subset \mathbb{R}^n$ with parameter $s > 0$ is then given by

$$\rho_s(\mathcal{L}) := \sum_{\boldsymbol{y} \in \mathcal{L}} \rho_s(\boldsymbol{y}) \ .$$

The *discrete Gaussian distribution* $D_{\mathcal{L},s}$ is the distribution over $\mathcal{L}$ induced by this measure, i.e., for any $\boldsymbol{y} \in \mathcal{L}$,

$$\Pr_{\boldsymbol{X} \sim D_{\mathcal{L},s}}[\boldsymbol{X} = \boldsymbol{y}] = \rho_s(\boldsymbol{y})/\rho_s(\mathcal{L}) \ .$$

We will need the following theorem from [BLP+13], which is a slight strengthening of a result in [GPV08].

Theorem 2.2. *There is an efficient algorithm that takes as input a basis* $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{R}^{n \times n}$ *for a lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and a parameter* $s \geq \sqrt{\log(2n+4)/\pi} \cdot \max_i \|\boldsymbol{b}_i\|$ *and outputs a sample from* $D_{\mathcal{L},s}$*.*

For $\varepsilon > 0$, the *smoothing parameter* of a lattice $\mathcal{L} \subset \mathbb{R}^n$ is the unique parameter $s > 0$ such that

$$\rho_{1/s}(\mathcal{L}^*) = 1 + \varepsilon \ .$$

Lemma 2.3 ([MR07, Lemma 4.1])**.** *For any lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and parameter* $s > \eta_\varepsilon(\mathcal{L})$ *for some* $\varepsilon \in (0,1)$*, if* $\boldsymbol{X} \sim D_s^n$*, then* $\boldsymbol{X} \bmod \mathcal{L}$ *is within statistical distance* $\varepsilon/2$ *of the uniform distribution modulo* $\mathcal{L}$*.*

Lemma 2.4 ([MR07, Lemma 3.2])**.** *For any lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and any* $\varepsilon > 2^{-n}$

$$\eta_\varepsilon(\mathcal{L}) \leq \sqrt{n}/\lambda_1(\mathcal{L}^*) \ .$$

We say that $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \in \mathcal{L}$ *generate a lattice* $\mathcal{L}$ if $\mathcal{L} = \{z_1 \boldsymbol{y}_1 + \cdots + z_m \boldsymbol{y}_m \ : \ z_i \in \mathbb{Z}\}$. In particular, when $m = n$, a generating set is simply a basis. We will need the following result due to Haviv and Regev [HR14], here applied for $\mathcal{L} = \mathbb{Z}^n$ for simplicity. (The more general result works for lattices with determinant one and parameters s such that the lattice has a basis consisting of vectors with length at most s.)

Lemma 2.5 ([HR14, Lemma 5.4])**.** *For any* $s \geq 1$ *and* $m \geq n^2 + n\log(s\sqrt{n})(n + 20\log\log(s\sqrt{n}))$*, if* $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \sim D_{\mathbb{Z}^n,s}$ *are sampled independently from* $D_{\mathbb{Z}^n,s}$*, then* $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m$ *is a generating set except with probability* $2^{-\Omega(n)}$*.*

2.3 Lattice problems

Definition 2.6. For $\gamma = \gamma(n) \geq 1$, the γ-approximate Shortest Vector Problem (γ-SVP) is the search problem defined as follows. Given a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ of a lattice $\mathcal{L}$ as input, output a non-zero vector $\boldsymbol{v} \in \mathcal{L}$ with $\|\boldsymbol{v}\| \leq \gamma \cdot \lambda_1(\mathcal{L})$.

Definition 2.7. For $\gamma = \gamma(n) \geq 1$, the unique Shortest Vector Problem with gap γ (γ-uSVP) is the search problem defined as follows. Given a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ of a lattice $\mathcal{L}$ satisfying $\gamma \cdot \lambda_1(\mathcal{L}) < \lambda_2(\mathcal{L})$ as input, output a vector $\boldsymbol{v} \in \mathcal{L}$ with $\|\boldsymbol{v}\| = \lambda_1(\mathcal{L})$.

We note that there is an almost trivial dimension-preserving reduction from γ-uSVP to γ-SVP. (Here and throughout the paper, we use *dimension-preserving* to mean that the reduction maps problem instances of dimension n to problem instances of dimension n.)

Fact 2.8. *For every* $\gamma \geq 1$*, there is a dimension-preserving Cook reduction from* γ-uSVP *to* γ-SVP*.*

Proof. Given an instance $\mathbf{B} \in \mathbb{Q}^{n \times n}$ of γ-uSVP as input, the reduction calls the γ-SVP oracle on $\mathbf{B}$, receiving a vector $\boldsymbol{v} = B\boldsymbol{x}$ as output for some coefficient vector $\boldsymbol{x} = (x_1, \ldots, x_n)$. The reduction returns $\boldsymbol{v}/\gcd(x_1, \ldots, x_n)$. It is clear that this reduction is efficient. It is correct because $0 < \|\boldsymbol{v}\| \le \gamma \cdot \lambda_1(\mathcal{L}(B)) < \lambda_2(\mathcal{L}(B))$, and so $\boldsymbol{v}$ must be a non-zero integer multiple of the unique (up to sign) shortest vector in $\mathcal{L}(B)$. $\square$

We next note the existence of useful approximation algorithm for SVP. Namely, the following result, due to Liu, Wang, Xu, and Zheng [LWXZ11], gives an algorithm for γ-SVP for constant γ that is substantially faster than the fastest known algorithm for exact SVP, which runs in $2^{n+o(n)}$ time [ADRS15]. See also [WLW15, AUV19, EV20]. In particular, [EV20, Theorem 3.2], gives a cleanly-stated generalization of the theorem below.

Theorem 2.9 ([LWXZ11])**.** *For every constant $\varepsilon > 0$ there exists a constant $\gamma = \gamma(\varepsilon) \ge 1$ depending only on ε such that there is a randomized algorithm that solves γ-SVP on lattices of dimension n in $2^{(0.802+\varepsilon)n} \cdot \mathrm{poly}(n)$ time.*

Let $T_{\mathrm{uSVP}}(\gamma, n)$ and $T_{\mathrm{SVP}}(\gamma, n)$ denote the fastest runtimes of (possibly randomized) algorithms for γ-uSVP and γ-SVP on lattices of dimension n. Then Fact 2.8 and Theorem 2.9 together show that for every constant $\varepsilon > 0$ there exists a constant $\gamma \ge 1$ such that

$$T_{\mathrm{uSVP}}(\gamma, n) \le T_{\mathrm{SVP}}(\gamma, n) \le 2^{(0.802+\varepsilon)n} \cdot \mathrm{poly}(n)\ . \tag{2}$$

2.3.1 Lattice problems on rotations of $\mathbb{Z}^n$

We say that two lattices $\mathcal{L}_1$, $\mathcal{L}_2$ of dimension n are *isomorphic*, which we denote by $\mathcal{L}_1 \cong \mathcal{L}_2$, if there exists $R \in \mathsf{O}_n(\mathbb{R})$ such that $R(\mathcal{L}_1) = \mathcal{L}_2$. We call lattices $\mathcal{L}$ satisfying $\mathcal{L} \cong \mathbb{Z}^n$ "rotations of $\mathbb{Z}^n$." We define γ-$\mathbb{Z}$SVP to be γ-SVP (as defined in Definition 2.6) with the additional requirement that the input basis $\mathbf{B}$ satisfy $\mathcal{L}(\mathbf{B}) \cong \mathbb{Z}^n$.

Definition 2.10. For $\gamma = \gamma(n) \ge 1$, the γ-approximate Shortest Vector Problem on rotations of $\mathbb{Z}^n$ (γ-$\mathbb{Z}$SVP) is the search problem defined as follows. Given a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ of a lattice $\mathcal{L}$ satisfying $\mathcal{L} \cong \mathbb{Z}^n$ as input, output a non-zero vector $\boldsymbol{v} \in \mathcal{L}$ with $\|\boldsymbol{v}\| \le \gamma \cdot \lambda_1(\mathcal{L})$.

When $\gamma = 1$, we simply write γ-$\mathbb{Z}$SVP as $\mathbb{Z}$SVP.

We also define the problem of *recovering* a rotation of $\mathbb{Z}^n$, i.e., of recovering an orthonormal basis of a lattice $\mathcal{L} \cong \mathbb{Z}^n$. Because this problem is equivalent to the search version of the so-called Lattice Isomorphism Problem when one lattice is fixed to be $\mathbb{Z}^n$, we call it $\mathbb{Z}$LIP.

Definition 2.11. The Lattice Isomorphism Problem on rotations of $\mathbb{Z}^n$ ($\mathbb{Z}$LIP) is the promise problem defined as follows. Given a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ of a lattice $\mathcal{L} \cong \mathbb{Z}^n$ as input, output $R \in \mathsf{O}_n(\mathbb{R})$ such that $R(\mathbb{Z}^n) = \mathcal{L}(B)$.

We conclude this section by noting that $\mathbb{Z}$SVP and $\mathbb{Z}$LIP are polynomial-time equivalent under dimension-preserving Cook reductions, and as such we will generally work with $\mathbb{Z}$SVP, which is somewhat simpler. (We note in passing that such a relationship is unknown and unlikely to hold for the corresponding problems SVP and LIP on general lattices.)

Fact 2.12. *The problems $\mathbb{Z}$SVP and $\mathbb{Z}$LIP are polynomial-time equivalent under dimension-preserving Cook reductions.*

Proof. The reduction from $\mathbb{Z}$SVP to $\mathbb{Z}$LIP works as follows. It calls its oracle for $\mathbb{Z}$LIP on the input instance of $\mathbb{Z}$SVP, receiving as output a matrix $R \in \mathsf{O}_n(\mathbb{R})$. It then returns the first column of R. Efficiency and correctness are both immediate.

We show that a Cook reduction from $\mathbb{Z}$LIP to $\mathbb{Z}$SVP exists by induction on n. Specifically, we prove that there exists such a reduction that, on input an instance $\mathbf{B}$ of $\mathbb{Z}$LIP of dimension n, outputs n pairwise-orthogonal, unit-length vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$. The base case of $n = 1$ is trivial. For $n > 1$, the reduction calls its $\mathbb{Z}$SVP oracle on the input instance of $\mathbb{Z}$LIP, receiving as output a vector $\boldsymbol{v}_1$ with $\|\boldsymbol{v}_1\| = 1$. It then recurses on (a basis of) the orthogonal projection $\pi_{\mathrm{span}(\boldsymbol{v}_1)^\perp}(\mathcal{L})$ of $\mathcal{L}$ onto $\mathrm{span}(\boldsymbol{v}_1)^\perp$, receiving as output $n-1$ vectors $\boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$. Finally, it outputs $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)$. The reduction is efficient because computing a basis of $\pi_{\mathrm{span}(\boldsymbol{v}_1)^\perp}(\mathcal{L})$ given $\mathbf{B}$ and $\boldsymbol{v}_1$ is efficient. Correctness follows by noting that $\pi_{\mathrm{span}(\boldsymbol{v}_1)^\perp}(\mathcal{L}) \cong \mathbb{Z}^{n-1}$, $\boldsymbol{v}_1 \perp \mathrm{span}(\boldsymbol{v}_2, \ldots, \boldsymbol{v}_n)$, and by the induction hypothesis. $\square$

2.4 Vector counting and lattice sparsification

Given a lattice $\mathcal{L}$, a vector $\boldsymbol{x} \in \mathcal{L}$ is called *primitive* if $\boldsymbol{x} \notin a\mathcal{L}$ for any integer $a > 1$. Note that $\mathbf{0}$ is not primitive regardless of $\mathcal{L}$. Let $\mathcal{L}_{\mathrm{prim}}$ denote the set of primitive vectors in $\mathcal{L}$. For a lattice $\mathcal{L}$ and $r > 0$, let $N(\mathcal{L}, r) := |\{\boldsymbol{x} \in \mathcal{L} : \|\boldsymbol{x}\| \le r\}|$ and let $N_{\mathrm{prim}}(\mathcal{L}, r) := |\{\boldsymbol{x} \in \mathcal{L}_{\mathrm{prim}} : \|\boldsymbol{x}\| \le r\}|/2$, where the latter expression counts primitive $\pm\boldsymbol{x}$ as a single vector.

We next state the main sparsification result that we will use.

Theorem 2.13 ([Ste16a, Theorem 4.1]). *For any lattice $\mathcal{L} \subseteq \mathbb{R}^n$ with basis B, $r > 0$, primitive lattice vectors $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathcal{L}_{prim}$ satisfying $\|\boldsymbol{x}_i\| \le r$ for all i and $\boldsymbol{x}_i \ne \pm\boldsymbol{x}_0$ for all $i > 0$, and prime $p \ge 101$, if $N_{prim}(\mathcal{L}, r) \le p/(20 \log p)$ then*

$$\frac{1}{p} - \frac{N}{p^2} \le \Pr[\langle \boldsymbol{a}, B^{-1}\boldsymbol{x}_0 \rangle \equiv 0 \pmod{p} \text{ and } \langle \boldsymbol{a}, B^{-1}\boldsymbol{x}_i \rangle \not\equiv 0 \pmod{p} \;\; \forall i > 0] \le \frac{1}{p} \ ,$$

where $\boldsymbol{a} \in \mathbb{Z}_p^n$ is chosen uniformly at random.

We will use the following upper bound from [Ste17] on the number of integer points in $\mathbb{Z}^n \cap r\mathcal{B}_2^n$ for various r, where $\mathcal{B}_2^n$ denotes the closed Euclidean unit ball. In fact, there are quite tight bounds of this form, as shown in [MO90].

Proposition 2.14 ([Ste17, Proof of Proposition 2.8.2]). *For any $n \ge 1$ and any radius $1 \le r \le \sqrt{n/2}$ with $r^2 \in \mathbb{Z}$,*

$$|\mathbb{Z}^n \cap r\mathcal{B}_2^n| \le (2ne^{1+s/2}/r^2)^{r^2} \le (20n/r^2)^{r^2} \ ,$$

where $s := \sqrt{\pi/\log(2n/r^2)}$.

2.5 Probability

Lemma 2.15 (Chernoff-Hoeffding bound [Hoe63]). *Let $X_1, \ldots, X_M \in [0,1]$ be independent and identically distributed random variables. Then, for $s > 0$,*

$$\Pr\Big[\Big|M\,\mathbb{E}[X_i] - \sum X_i\Big| \ge sM\Big] \le 2e^{-Ms^2/10} \ .$$

2.6 On bit lengths, input formats, and representing real numbers

Throughout this work, we adopt the common convention of expressing the running times of lattice algorithms in terms of the dimension n only, ignoring any dependence on the bit length ℓ of the entries of the input matrix. Formally, we should specify a particular input format for the lattice basis (e.g., by restricting our attention to rational numbers and using the natural binary representation of a rational matrix, or by working with algebraic numbers represented by their minimal polynomials), and our running time should of course have some dependence on ℓ. Consideration of the bit length would simply add a $\mathrm{poly}(\ell)$ factor to the running time for the algorithms and reductions considered in this paper for any reasonable input format.

Similarly, our reductions sometimes apply random orthogonal linear transformations $R \sim \mathsf{O}_n(\mathbb{R})$, without worrying about how we represent such a linear transformation. There are at least two solutions to this problem: one can either use a suitable discretization of $\mathsf{O}_n(\mathbb{R})$, or one can simply switch from working directly with a basis $\mathbf{B}$ to working with the associated Gram matrix $\boldsymbol{G} := \mathbf{B}^T\mathbf{B}$ or some canonical rotation of $\mathbf{B}$ such as the QR-decomposition, as discussed in Section 1.3.

3 How to sample a provably secure basis

In this section, we show how to sample a basis $\mathbf{B}$ for a rotation of $\mathbb{Z}^n$ that is "provably at least as secure as any other basis." In particular, we show a distribution of bases $\mathbf{B}$ of rotations of $\mathbb{Z}^n$ that can be sampled efficiently given any basis of a rotation of $\mathbb{Z}^n$ in such a way that together with the orthogonal transformation R mapping the original lattice to the new lattice. This implies that "if a computational problem can be solved efficiently given a basis from this distribution, then it can be solved efficiently given any basis." Similar ideas appeared in [HR14, DvW21a].

In fact, we give a class of distributions, one for each efficient *rotation-invariant* algorithm that converts a generating set to a basis. Such an algorithm $\mathcal{A}$ takes as input vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \in \mathcal{L}$ that form a generating set of a lattice $\mathcal{L}$ and outputs a basis $\mathbf{B}$ of $\mathcal{L}$, with the property that for any orthogonal transformation $R \in \mathsf{O}_n(\mathbb{R})$, $\mathcal{A}(R\boldsymbol{y}_1, \ldots, R\boldsymbol{y}_m) = R\mathcal{A}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m)$. (Here, for simplicity, we are assuming that $\mathcal{A}$ is a deterministic algorithm. We could generalize this definition to randomized algorithms and simply require that the distribution of $\mathcal{A}(R\boldsymbol{y}_1, \ldots, R\boldsymbol{y}_m)$ be statistically close to the distribution of $R\mathcal{A}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m)$.) One can equivalently consider algorithms that work with the gram matrix $\boldsymbol{G} \in \mathbb{R}^{m \times m}$ of the generating set, given by $G_{i,j} := \langle \boldsymbol{y}_i, \boldsymbol{y}_j \rangle$, which is invariant under rotations. For example, the LLL algorithm yields an efficient rotation-invariant algorithm that converts a generating set to a basis.

Given such an $\mathcal{A}$, our distribution is then the following.

Definition 3.1. For any efficient rotation-invariant algorithm $\mathcal{A}$ that converts a generating set to a basis and parameter $s = s(n) \geq 1$ the distribution $(\mathcal{A}, s)$-$\mathbb{Z}$DGS is sampled as follows. For $i = 1, 2, 3, \ldots$, sample $\boldsymbol{z}_i \sim D_{\mathbb{Z}^n, s}$. Let $\mathbf{B} := \mathcal{A}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_i)$. If $\mathbf{B} \in \mathbb{R}^{n \times n}$ is full rank and $|\det(\mathbf{B})| = 1$, then sample a uniformly random orthogonal matrix $R \sim \mathsf{O}_n(\mathbb{R})$ and output $\mathbf{B}' := R\mathbf{B}$. Otherwise, continue the loop.

Notice that the resulting basis is in fact a basis of a rotation of $\mathbb{Z}^n$, specifically, $R\mathbb{Z}^n$. By Lemma 2.5, the above procedure terminates in polynomial time except with negligible probability.[3]

Theorem 3.2. *For any efficient rotation-invariant algorithm $\mathcal{A}$ that converts a generating set into a basis, there is an efficient randomized algorithm that takes as input a basis $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{R}^{n \times n}$ for a rotation $\mathcal{L}$ of $\mathbb{Z}^n$ and a parameter $s \geq \sqrt{\log(2n+4)/\pi} \cdot \max \|\boldsymbol{b}_i\|$ and outputs a basis $\mathbf{B}' \in \mathbb{R}^{n \times n}$ generating $\mathcal{L}'$ that is distributed exactly as $(\mathcal{A}, s)$-$\mathbb{Z}$DGS together with an orthogonal transformation $R \in \mathsf{O}_n(\mathbb{R})$ such that $R\mathcal{L} = \mathcal{L}'$.*

Proof. The algorithm behaves as follows. For $i = 1, 2, 3, \ldots$, the algorithm uses the procedure from Theorem 2.2 to sample $\boldsymbol{y}_i \sim D_{\mathcal{L}, s}$, where $\mathcal{L}$ is the lattice generated by $\mathbf{B}$. It then computes $\mathbf{B}^\dagger := \mathcal{A}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_i)$. If the lattice generated by $\mathbf{B}^\dagger$ has full rank and determinant one, then the algorithm outputs $\mathbf{B}' := R\mathbf{B}^\dagger$ and R, where $R \sim \mathsf{O}_n(\mathbb{R})$ is a uniformly random rotation. Otherwise, it continues.

To see why this is correct, let $R' \in \mathsf{O}_n(\mathbb{R})$ be an orthogonal transformation such that $\mathbb{Z}^n = R'\mathcal{L}$. Let $\boldsymbol{y}'_i := R'\boldsymbol{y}_i$, and notice that the $\boldsymbol{y}'_i$ are distributed as independent samples from $D_{\mathbb{Z}^n, s}$. It follows from the fact that $\mathcal{A}$ is rotation invariant that $R'\mathbf{B}^\dagger = \mathcal{A}(\boldsymbol{y}'_1, \ldots, \boldsymbol{y}'_i)$. Clearly $\mathbf{B}^\dagger$ is full rank and has determinant one if and only if $R'\mathbf{B}^\dagger$ has this same property. Therefore, $\mathbf{B}'$ is distributed exactly as $R(R')^{-1}\mathcal{A}(\boldsymbol{y}'_1, \ldots, \boldsymbol{y}'_i)$ (conditioned on the rank and determinant conditions being satisfied). Since R is a uniformly random orthogonal transformation, this is distributed identically to $R''\mathcal{A}(\boldsymbol{y}'_1, \ldots, \boldsymbol{y}'_i)$ for $R'' \sim \mathsf{O}_n(\mathbb{R})$. Notice that this is exactly the $\mathbb{Z}$DGS distribution.

Finally, as we observed above, Lemma 2.5 implies that after $\mathrm{poly}(n, \log s)$ samples, $\boldsymbol{y}'_1, \ldots, \boldsymbol{y}'_i$ will generate $\mathbb{Z}^n$ with high probability, in which case $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_i$ will generate $\mathcal{L}$. Therefore, the algorithm terminates in polynomial time (with high probability). $\square$

The following corollary shows that we can achieve the same result for a fixed parameter s (regardless of the length of the input basis).

[3]One can also use an alternative optimized sampling procedure that "runs the algorithm $\mathcal{A}$ iteratively." I.e., we can maintain a running basis $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k)$, which starts as the empty basis. And, after sampling $\boldsymbol{y}_i$, we can simply update $\mathbf{B}$ to $\mathcal{A}(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k, \boldsymbol{y}_i)$, continuing in this fashion until $k = n$ and $\det(\mathbf{B}) = \pm 1$. E.g., if $\mathcal{A}$ is the LLL algorithm, then this will significantly improve performance in practice.

Corollary 3.3. *For any efficient rotation-invariant algorithm $\mathcal{A}$ that converts a generating set into a basis, there is an efficient randomized algorithm that takes as input any basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ for a rotation $\mathcal{L}$ of $\mathbb{Z}^n$ and outputs a basis $\mathbf{B}' \in \mathbb{R}^{n \times n}$ generating $\mathcal{L}'$ and rotation R such that $\mathbf{B}'$ is distributed as $(\mathcal{A}, s)$-$\mathbb{Z}$DGS and $R\mathcal{L} = \mathcal{L}'$, where $s = 2^n$.*

Proof. The algorithm simply runs the LLL algorithm on $\mathbf{B}$, receiving as output some basis $\mathbf{B}^\dagger = (\boldsymbol{b}_1^\dagger, \ldots, \boldsymbol{b}_n^\dagger)$ for $\mathcal{L}$ with $\|\boldsymbol{b}_i^\dagger\| \leq 2^{n/2}$. It then runs the procedure from Theorem 3.2 and outputs the result. □

Using Corollary 3.3, we can easily reduce worst-case variants of lattice problems on $\mathbb{Z}^n$ to variants in which the input basis is sampled from $\mathbb{Z}$DGS. As an example, we show a random self-reduction for SVP over rotations of $\mathbb{Z}^n$ below. (We also use this idea in Section 4.)

Definition 3.4. For any $\gamma = \gamma(n) \geq 1$ and any efficient rotation-invariant algorithm $\mathcal{A}$, the $(\mathcal{A}, \gamma)$-ac$\mathbb{Z}$SVP problem is defined as follows. The input is a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ sampled from $(\mathcal{A}, 2^n)$-$\mathbb{Z}$DGS generating a rotation $\mathcal{L}$ of $\mathbb{Z}^n$. The goal is to output $\boldsymbol{y} \in \mathcal{L}$ with $0 < \|\boldsymbol{y}\| \leq \gamma$.

Theorem 3.5. *For any efficient rotation-invariant algorithm $\mathcal{A}$ and any $\gamma \geq 1$, there is an efficient reduction from γ-$\mathbb{Z}$SVP to $(\mathcal{A}, \gamma)$-ac$\mathbb{Z}$SVP.*

Proof. The reduction takes as input a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ for a rotation $\mathcal{L}$ of $\mathbb{Z}^n$ and simply runs the procedure from Corollary 3.3, receiving as output a basis $\mathbf{B}'$ sampled from $(\mathcal{A}, 2^n)$-$\mathbb{Z}$DGS generating $\mathcal{L}'$ together with a rotation R such that $R\mathcal{L} = \mathcal{L}'$. It then calls its $(\mathcal{A}, \gamma)$-ac$\mathbb{Z}$SVP on input $\mathbf{B}'$, receiving as output some vector $\boldsymbol{y}' \in \mathcal{L}'$. Finally, it outputs $\boldsymbol{y} := R^{-1}\boldsymbol{y}'$. □

4 We have an encryption scheme to sell you

We now consider the possibility that it actually is "hard to recognize $\mathbb{Z}^n$" (where we must formalize what this means rather carefully), and we show that this implies the existence of a relatively simple public-key encryption scheme.

The encryption scheme itself is described below. There are public parameters $s > 0$, $r > 0$, and $d > 0$, which are all functions of the security parameter n (i.e., $s = s(n)$, $r = r(n)$, and $d = d(n)$), as well as an algorithm $\mathcal{A}$ which is an efficient rotation-invariant algorithm for converting a generating set to a basis, as in Definition 3.1. In particular, the parameter s will control the length of the basis used as the public key, the parameter r is a noise parameter (which we will take as large as possible, roughly $r \approx \sqrt{\log n}$ while still allowing for decryption), and the parameter $d \approx n/12 \cdot (1 - e^{-r^2})$ is a distance threshold that the decryption algorithm will use to distinguish encryptions of zero from encryptions of one.

- $\mathsf{Gen}(1^n)$: Sample vectors $\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \ldots$ independently from $D_{\mathbb{Z}^n,s}$ until $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ generate $\mathbb{Z}^n$. Let $\mathbf{B} := \mathcal{A}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k)$ and let $\boldsymbol{G} := \mathbf{B}^T\mathbf{B}$. Output $sk := \mathbf{B}$ and $pk := \boldsymbol{G}$.
- $\mathsf{Enc}(pk, b \in \{0, 1\})$: If $b = 0$, sample $\boldsymbol{X}$ from a continuous Gaussian distribution with covariance matrix $r^2\boldsymbol{G}^{-1}$,[4] and output $\boldsymbol{c} := \boldsymbol{X} \bmod 1$ (i.e., the coordinates of $\boldsymbol{c}$ are the fractional parts of the coordinates of $\boldsymbol{X}$). Otherwise, output uniformly random $\boldsymbol{c} \sim [0, 1)^n$.
- $\mathsf{Dec}(sk, \boldsymbol{c})$: Set $\boldsymbol{t} = (t_1, \ldots, t_n)^T := \mathbf{B}\boldsymbol{c}$. Output 1 if $\sum(t_i - \lfloor t_i \rceil)^2 > d$ and 0 otherwise.

We first concern ourselves with the correctness of this scheme. In particular, the following lemma tells us that the decryption will answer correctly except with probability roughly $\exp(-e^{-\pi r^2}n)$. For security, we will want to take r to be as big as possible, so we will take r to be slightly smaller than $\sqrt{(\log n - \log\log n)/\pi}$,

[4] In other words, $\boldsymbol{X}$ is sampled from the probability distribution with probability density function

$$r^n \det(\boldsymbol{G})^{-1/2} \cdot \exp(-\pi \boldsymbol{x}^T \boldsymbol{G} \boldsymbol{x}/r^2) = \frac{r^n}{\det(\mathbf{B})} \cdot \exp(-\pi\|\mathbf{B}\boldsymbol{x}\|^2/r^2) \ .$$

Notice that we would get an identical distribution if we were to sample $\boldsymbol{Y} \sim D_r^n$ and set $\boldsymbol{X} := \mathbf{B}^{-1}\boldsymbol{Y}$.

in order to make this failure probability negligible. E.g., we can take $r = \sqrt{(\log n - (\log\log n)^2)/\pi}$. This is essentially the maximal choice for r that still maintains correctness, since if we took, e.g, $r \gtrsim \sqrt{\log(4n)/\pi}$, then ciphertexts of zero would be within statistical distance $1/2$ of ciphertexts of one, making decryption failures unreasonably common.

Lemma 4.1. *For $r \ge 1$, let $\delta := \exp(-\pi r^2)$ and*

$$d := \frac{n}{12} - \frac{\delta}{4\pi^2} \cdot n \; .$$

Then, the decryption algorithm described above outputs the correct bit b except with probability at most

$$2\exp(-c\delta^2 n) \; .$$

for some constant $c > 0$.

Proof. For the case $b = 1$, we simply notice that $\boldsymbol{t}$ is uniformly random in a fundamental domain of $\mathbb{Z}^n$. It follows that $t_i - \lfloor t_i \rceil$ is uniformly random in the interval $[-1/2, 1/2)$ and independent of the other coordinates. In particular $\mathbb{E}[(t_i - \lfloor t_i \rceil)^2] = 1/12$. It then follows from the Chernoff-Hoeffding bound (Lemma 2.15) that

$$\Pr\Big[\sum (t_i - \lfloor t_i \rceil)^2 \le d\Big] \le \exp(-\delta^2 n/1000)$$

We now consider the case $b = 0$. Write $\boldsymbol{c} = \boldsymbol{X} + \boldsymbol{z}$ for $\boldsymbol{z} \in \mathbb{Z}^n$. Then, $\boldsymbol{t} = \mathbf{B}\boldsymbol{c} = \mathbf{B}\boldsymbol{X} + \mathbf{B}\boldsymbol{z} = \mathbf{B}\boldsymbol{X} \bmod 1$. (Here, we crucially rely on the fact that $\mathbf{B}$ is an integer matrix.) Notice that $\mathbf{B}\boldsymbol{X}$ is distributed exactly as a continuous Gaussian with covariance $\mathbf{B}(r^2\boldsymbol{G}^{-1})\mathbf{B}^T = r^2$, i.e., as D_r^n. Therefore, $\sum (t_i - \lfloor t_i \rceil)^2$ is distributed identically to $\mathrm{dist}(\boldsymbol{Y}, \mathbb{Z}^n)^2$, where $\boldsymbol{Y} \sim D_r^n$. By Lemma 2.1,

$$\Pr[\mathrm{dist}(\boldsymbol{Y}, \mathbb{Z}^n)^2 > d] \le 2\exp(-(d - \nu - \varepsilon n)^2/10) \; ,$$

where

$$\nu := \frac{n}{12} - \frac{\delta}{2\pi^2} \cdot n \; ,$$

and $\varepsilon := \delta^4/3$. Notice that

$$\frac{d - \nu - \varepsilon n}{n} = \frac{\delta}{4\pi^2} - \delta^4/3 > \delta/100 \; .$$

The result follows. ☐

4.1 Basic security

We now observe that the above scheme is semantically secure if (and only if) the following problem is hard. The only distinction between this problem and the problem of breaking the semantic security of the encryption scheme is that in the problem below the underlying lattice is specified by a worst-case basis $\mathbf{B}$ instead of an average-case Gram matrix $\boldsymbol{G}$. We will reduce between the two problems using the ideas from Section 3.

(Here and below, we have an additional parameter ρ, which is a bound on the lengths of the input basis vectors. If we set $s = 2^n$ in our encryption scheme, then we could remove ρ by using the LLL algorithm, as we did in Section 3. However, we choose to keep the parameter ρ to allow for the possibility of smaller choices of s, which yield smaller entries in the Gram matrix $\boldsymbol{G}$ and therefore a smaller public key.)

Definition 4.2. For parameters $\rho = \rho(n) > 0$ and $r = r(n) > 0$, the (ρ, r)-$\mathbb{Z}$GvU problem (Gaussian versus Uniform mod $\mathbb{Z}^n$) is the promise problem defined as follows. The input is a basis $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{R}^{n \times n}$ such that $\|\boldsymbol{b}_i\| \le \rho$ that generates a rotation of $\mathbb{Z}^n$, and a vector $\boldsymbol{y} \in [0,1)^n$, where $\boldsymbol{y}$ is sampled as follows. A bit $b \sim \{0,1\}$ is sampled uniformly at random. If $b = 0$, $\boldsymbol{y} = \mathbf{B}^{-1}\boldsymbol{X} \bmod 1$ for $\boldsymbol{X} \sim D_r$, and if $b = 1$, $\boldsymbol{y} \sim [0,1)^n$. The goal is to output b.

We say that (ρ, r)-$\mathbb{Z}$*GvU is hard* if no probabilistic polynomial-time algorithm $\mathcal{A}$ can solve this problem with probability better than $1/2 + \mathrm{negl}(n)$.

Theorem 4.3. *If (ρ, r)-$\mathbb{Z}$GvU is hard for some ρ, r, then the above encryption scheme is semantically secure with parameters $s := \sqrt{\log(2n+4)/\pi} \cdot \rho$ and r.*

Proof. Suppose that there is a probabilistic polynomial-time adversary $\mathcal{B}$ that has non-negligible advantage in breaking the semantic security of the encryption scheme. We construct an efficient algorithm $\mathcal{E}$ that solves $\mathbb{Z}$GvU with probability non-negligibly larger than 1/2.

The algorithm $\mathcal{E}$ takes as input a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ generating a lattice $\mathcal{L}$, and $\boldsymbol{y} \in [0,1)^n$. It then uses the procedure from Theorem 3.2 (using the same algorithm $\mathcal{A}$ used in the cryptosystem) to convert this into a basis $\mathbf{B}'$ for a rotation of $\mathcal{L}$ and sets $\boldsymbol{G} := (\mathbf{B}')^T \mathbf{B}'$. It then sets $\boldsymbol{c} := (\mathbf{B}')^{-1}\mathbf{B}\boldsymbol{y} \bmod 1$. Finally, $\mathcal{E}$ calls $\mathcal{B}$ on input $\boldsymbol{G}$ and $\boldsymbol{c}$ and outputs whatever $\mathcal{B}$ outputs.

It is clear that $\mathcal{E}$ is efficient. Furthermore, if $\boldsymbol{y}$ is uniformly random modulo 1, then clearly $\boldsymbol{c}$ is also uniformly random modulo 1. On the other hand, if $\boldsymbol{y} = \mathbf{B}^{-1}\boldsymbol{X} \bmod 1$ for $\boldsymbol{X} \sim D_r$, then

$$\boldsymbol{c} = (\mathbf{B}')^{-1}\mathbf{B}\boldsymbol{y} \bmod 1 = (\mathbf{B}')^{-1}\boldsymbol{X} \bmod 1 \ .$$

Notice that $(\mathbf{B}')^{-1}\boldsymbol{X}$ is distributed exactly as a Gaussian with covariance $r^2\boldsymbol{G}^{-1}$. Therefore, when $b = 0$, $\boldsymbol{c}$ is distributed exactly like an encryption of zero, and when $b = 1$, $\boldsymbol{c}$ is distributed exactly like an encryption of one. □

4.2 A worst-case to average-case reduction (of a sort)

Of course, $\mathbb{Z}$GvU is a rather artificial problem. Below, we show reductions to it from worst-case problems that ask us to distinguish $\mathbb{Z}^n$ from a lattice that is different from $\mathbb{Z}^n$ in a specific way. These can be thought of as "$\mathbb{Z}^n$ versions" of the traditional worst-case lattice problems GapSPP and GapSVP.

Recall that $\eta_\varepsilon(\mathbb{Z}^n) \approx \sqrt{\log(2n/\varepsilon)/\pi}$ for small ε.

Definition 4.4. For any approximation factor $\alpha = \alpha(n) \geq 1$, $\varepsilon \in (0, 1/2)$, and a length bound $\rho = \rho(n) > 0$, the problem $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}\mathsf{GapSPP}$ is defined as follows. The input is a basis $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{R}^{n \times n}$ for a lattice $\mathcal{L}$ satisfying $\|\boldsymbol{b}_i\| \leq \rho$. The goal is to output YES if $\mathcal{L} \cong \mathbb{Z}^n$ and to output NO if $\eta_\varepsilon(\mathcal{L}) < \eta_\varepsilon(\mathbb{Z}^n)/\alpha$.

The below reduction shows that if $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}\mathsf{GapSPP}$ is hard, then our encryption scheme with

$$r := \sqrt{(\log n - (\log \log n)^2)/\pi}$$

is secure for any $\varepsilon < n^{-\omega(1)}$ and $\alpha \leq \eta_\varepsilon(\mathbb{Z}^n)/r \approx \sqrt{\log(n/\varepsilon)/\log n}$.

Theorem 4.5. *For any efficiently computable $\varepsilon = \varepsilon(n) \in (0,1)$ and integer $\ell = \ell(n) \geq 100n/(\delta - \varepsilon)^2$, there is a reduction from $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}\mathsf{GapSPP}$ to (ρ, r)-$\mathbb{Z}$GvU that runs in time $\mathrm{poly}(n) \cdot \ell$ and answers correctly except with probability at most 2^{-n}, where $\alpha := \eta_\varepsilon(\mathbb{Z}^n)/r$ and the success probability of the $\mathbb{Z}$GvU oracle is $1/2 + \delta$, provided that $\delta > \varepsilon$.*

In particular, if $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}$GapSPP is hard for any negligible $\varepsilon = \varepsilon(n) < n^{-\omega(1)}$, then (ρ, r)-$\mathbb{Z}$GvU is hard.

Proof. The reduction takes as input a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and behaves as follows. For $i = 1, \ldots, \ell$, it samples a uniformly random bit $b_i \sim \{0,1\}$. If $b_i = 0$, it samples $\boldsymbol{X}_i \sim D_r^n$ and sets $\boldsymbol{y}_i := \mathbf{B}^{-1}\boldsymbol{X}_i \bmod 1$, and if $b_i = 1$, it samples $\boldsymbol{y}_i \sim [0,1)^n$. It then calls the $\mathbb{Z}$GvU oracle on input $\mathbf{B}$ and $\boldsymbol{y}_i$, receiving as output some bit $b_i^* \in \{0,1\}$.

Let p be the fraction of indices i such that $b_i = b_i^*$. The algorithm outputs YES if $p \geq 1/2 + \varepsilon + \sqrt{20n/\ell}$. Otherwise, it outputs NO.

The running time is clear. To prove correctness, we first notice that in the YES case, the input to the $\mathbb{Z}$GvU oracle is distributed identically to the $\mathbb{Z}$GvU input. It follows that for each i, $\Pr[b_i^* = b_i] = 1/2 + \delta$. Furthermore, these events are independent. Therefore, by the Chernoff-Hoeffding bound (Lemma 2.15),

$$\Pr[p < 1/2 + \varepsilon + \sqrt{20n/\ell}] \leq 2\exp(-\ell(\delta - \varepsilon - \sqrt{20n/\ell})^2/10) \leq 2^{-n} \ ,$$

as needed.

On the other hand, in the NO case, by Lemma 2.3, $\boldsymbol{y}_i$ is within statistical distance ε of a uniformly random element in $[0,1)^n$. It follows that, regardless of the behavior of the oracle, for each i, $\Pr[b_i^* = b_i] \le 1/2 + \varepsilon$, and again these events are independent. Therefore, by the Chernoff-Hoeffding bound again,

$$\Pr[p \ge 1/2 + \varepsilon + \sqrt{20n/\ell}] \le 2\exp(-2n) \le 2^{-n} \ ,$$

as needed. $\square$

Notice that the following definition works with the dual lattice $\mathcal{L}^*$ for convenience.

Definition 4.6. For parameters $\rho = \rho(n) > 0$ and $\gamma = \gamma(n) \ge 1$, the problem (ρ, γ)-$\mathbb{Z}\mathsf{GapSVP}$ is defined as follows. The input is a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$. The goal is to output YES if $\mathcal{L} \cong \mathbb{Z}^n$ and to output NO if $\lambda_1(\mathcal{L}^*) > \gamma$.

Theorem 4.7. *For any $\varepsilon = \varepsilon(n)$ with $2^{-n} < \varepsilon < 1/2$, $\rho = \rho(n) > 0$, and $\gamma = \gamma(n) \ge 10\sqrt{n/\log(n/\varepsilon)}$, there is an efficient reduction from (ρ, γ)-$\mathbb{Z}\mathsf{GapSVP}$ to $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}\mathsf{GapSPP}$ for $\alpha := \gamma\sqrt{\log(n/\varepsilon)/n}/10$.*

Proof. The reduction simply calls its $\mathbb{Z}\mathsf{GapSPP}$ oracle on its input, and outputs whatever the oracle outputs. To see that this reduction is correct, it suffices to consider the NO case. Indeed, by Lemma 2.4 if $\lambda_1(\mathcal{L}^*) > \gamma$, then $\eta_\varepsilon(\mathcal{L}) < \sqrt{n}/\gamma \le 10\sqrt{n/\log(n/\varepsilon)} \cdot \eta_\varepsilon(\mathbb{Z}^n)/\gamma = \eta_\varepsilon(\mathbb{Z}^n)/\alpha$, so that the oracle must output NO. $\square$

4.3 Putting everything together

Finally, we put the reductions above together to obtain a correct public-key encryption scheme that is secure assuming that $\mathbb{Z}\mathsf{GapSVP}$ (or even $\mathbb{Z}\mathsf{GapSPP}$) is hard.

Theorem 4.8. *Let $r := \sqrt{(\log n - (\log\log n)^2)/\pi}$, and let d be as in Lemma 4.1. Then, the above encryption scheme is correct, and for any $s = s(n) > 0$ and any $2^{-n} < \varepsilon < n^{-\omega(1)}$ the scheme is secure* either *if $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}\mathsf{GapSPP}$ is hard for $\alpha := \eta_\varepsilon(\mathbb{Z}^n)/r \approx \sqrt{\log(n/\varepsilon)/\log n}$ and $\rho := s/\sqrt{(\log 2n + 4)/\pi}$* or *if (ρ, γ)-$\mathbb{Z}\mathsf{GapSVP}$ is hard for $\gamma := 10\sqrt{n/\log(n/\varepsilon)} \cdot \alpha \approx \sqrt{n/\log n}$.*

4.4 Is $\mathbb{Z}^n$ the *best* lattice for cryptography? (with a connection to reverse Minkowski theorems)

Much of the analysis that we did above could replace $\mathbb{Z}^n$ with a different lattice $\mathcal{L}$. Indeed, we are *not* willing to conjecture that it is hard to solve SVP on a rotation of $\mathbb{Z}^n$, so we are certainly *not* conjecturing that the encryption scheme described above is actually secure because we are not willing to conjecture that even $\mathbb{Z}$SVP is hard (let alone the variants described above).

Setting that gigantic caveat aside for the moment, we present an interesting argument that $\mathbb{Z}^n$ might actually be the *best* lattice for cryptographic purposes.

First, notice that we show security of our scheme assuming the hardness of $\mathbb{Z}$GapSVP with an approximation factor $\gamma \approx \sqrt{n/\log n}$. If this were a reduction from GapSVP, rather than the exotic problem $\mathbb{Z}$GapSVP, then this would be truly remarkable. Indeed, the best security results that we have for public-key encryption schemes still require hardness of γ-GapSVP for $\gamma \gtrsim n^{3/2}$. So, in some sense, we achieve a much better worst-case approximation factor than what is known for, e.g., LWE-based public-key encryption. (Of course, this is quite misleading, since $\mathbb{Z}$GapSVP is actually trivial for $\gamma \ge \sqrt{n}$ by Minkowski's theorem. So, one could also argue that the approximation factor $\gamma \approx \sqrt{n/\log n}$ is "only slightly better than trivial.")

The relationship with $\mathbb{Z}$GapSPP is more interesting. In particular, there is a certain precise sense in which $\mathbb{Z}^n$ "has the largest smoothing parameter of any lattice." Such a statement can be made formal in a reverse Minkowski theorem [DR16, RS17]. In particular, recent work [RS17, RS21] comes quite close to proving a statement of the form "any lattice $\mathcal{L} \subset \mathbb{R}^n$ with determinant one and an integral Gram matrix $\boldsymbol{G}$ has $\eta_\varepsilon(\mathcal{L}) \le \eta_\varepsilon(\mathbb{Z}^n)$, with equality if and only if $\mathcal{L} \cong \mathbb{Z}^n$."[5] Such a statement would suggest that (1) $\mathbb{Z}^n$ is the

[5]Such lattices are called *unimodular*. Specifically, [RS21] prove that $\eta_\varepsilon(\mathcal{L}) \le (2 + o(1))\eta_\varepsilon(\mathbb{Z}^n)$ for any unimodular lattice. Furthermore, [RS17] proves that $\eta_\varepsilon(\mathcal{L}) \le \eta_\varepsilon(\mathbb{Z}^n)$ with equality if and only if $\mathcal{L} \cong \mathbb{Z}^n$ for very small $\varepsilon \le 2^{-Cn}$, and [ERS21] proves a similar statement for a non-Gaussian test function.

best lattice (among those with integral Gram matrices) for the purposes of decoding (since taking r above smoothing makes decoding impossible by definition); and (2) that solving $\mathbb{Z}$GapSPP might be more-or-less as hard as simply recognizing a rotation of $\mathbb{Z}^n$. In particular, it is trivial to check that the input lattice has an integral Gram matrix and determinant one, so recognizing $\mathbb{Z}^n$ is equivalent to distinguishing $\mathbb{Z}^n$ from any other such lattice. If all such lattices have smaller smoothing parameter than $\mathbb{Z}^n$, then distinguishing $\mathbb{Z}^n$ from a lattice with significantly smaller smoothing parameter is closely related to the problem of simply distinguishing $\mathbb{Z}^n$ from *any* lattice.

Of course, the above argument is not quantitative. There exist unimodular lattices that are distinct from $\mathbb{Z}^n$ with $\eta_\varepsilon(\mathcal{L}) > \eta_\varepsilon(\mathbb{Z}^n)/\alpha$ for rather small α, and we do *not* show that our encryption scheme is secure even under the assumption that it is hard to distinguish a rotation of $\mathbb{Z}^n$ from such a lattice $\mathcal{L}$. Instead, we are noting that α-$\mathbb{Z}$GapSPP amounts to the problem of "recognizing $\mathbb{Z}^n$ by identifying one of its most distinguishing features."

4.5 Concerning the genus of $\mathbb{Z}^n$

We note that there exist certain efficiently computable arithmetic lattice invariants (i.e., arithmetic properties of a lattice that are invariant under rotation) that can sometimes be used to determine that two lattices are not isomorphic. The equivalence class of lattices with the same arithmetic invariants is called a *genus*. The authors do not know whether there exist lattices $\mathcal{L}$ in the genus of $\mathbb{Z}^n$ that have $\lambda_1(\mathcal{L}) \gtrsim \sqrt{n/\log n}$, and it seems like proving the existence of such a lattice (or that they do not exist) might be difficult. If no such lattices exist, then $\mathbb{Z}$GapSVP can be solved efficiently for the parameters in Theorem 4.8 by simply checking whether the input lattice is in the same genus as $\mathbb{Z}^n$. It is, however, known that there exist lattices with very large minimum distance that share some of the simplest arithmetic invariants with $\mathbb{Z}^n$—specifically, there exist odd unimodular lattices with $\lambda_1(\mathcal{L}) \gtrsim (n/(2\pi e))^{n/2}$ [MH73, Theorem 9.5].

In fact, the authors do not even know if there exist lattices $\mathcal{L}$ in the same genus as $\mathbb{Z}^n$ with $\eta_\varepsilon(\mathcal{L}) \lesssim \eta_\varepsilon(\mathbb{Z}^n)/\sqrt{\log n} \approx \sqrt{\log(1/\varepsilon)/\log n}$ for $\varepsilon < n^{-\omega(1)}$. If such lattices do not exist, then $\mathbb{Z}$GapSPP can also be solved efficiently for the parameters in Theorem 4.8. However, given the discussion above and the rather small approximation factor of $\alpha \approx \sqrt{\log n}$, it seems likely that such lattices exist.

5 Reductions and provable algorithms

In this section, we give reductions from $\mathbb{Z}$SVP to *approximate* (unique-)SVP. In particular, our main result yields a randomized polynomial-time reduction from $\mathbb{Z}$SVP to γ-uSVP for any constant $\gamma \geq 1$. By combining this reduction with a known approximation algorithm for SVP, we show that for any constant $\varepsilon > 0$ there is a $(2^{(0.802+\varepsilon)n} \cdot \mathrm{poly}(n))$-time algorithm for $\mathbb{Z}$SVP.[6] This improves exponentially over the fastest known algorithm for SVP on general lattices [ADRS15], which runs in $2^{n+o(n)}$ time and was previously the fastest known algorithm even for the special case of $\mathbb{Z}$SVP.

Interpreted differently, our main reduction also shows conditional *hardness* of uSVP. Namely, if one were to hypothesize that there is no (possibly randomized) polynomial-time algorithm for $\mathbb{Z}$SVP, then it implies that there is no randomized polynomial-time algorithm for solving γ-uSVP for any constant $\gamma \geq 1$. This is notable because uSVP is not known to be NP-hard for any constant factor greater than 1. We also note that our main reduction generalizes to arbitrary lattices with few short vectors and may be of independent interest.

5.1 A simple projection-based reduction

Before giving our main reduction, we start with a simple reduction from $\mathbb{Z}$SVP to $\sqrt{2}$-SVP using a deterministic, "projection-based" approach. More specifically, we start by querying our $\sqrt{2}$-SVP oracle on the input lattice $\mathcal{L}$, and if the vector $\boldsymbol{v}$ that it returns does not have unit length then we recurse on the orthogonal

[6] We note again in passing that under a purely geometric conjecture we would in fact obtain a running time of $(4/3)^{n+o(n)} \approx 2^{0.415n}$ [Ste20].

projection $\pi_{\boldsymbol{v}^\perp}(\mathcal{L})$ of $\mathcal{L}$ onto $\boldsymbol{v}^\perp$. Although the results in Section 5.2 largely subsume it, we choose to present it because of its simplicity and the fact that the projection technique seems likely to generalize. (Indeed, we know similar reductions that work for $\sqrt{3}$-SVP and even 2-SVP, but our analysis of those involves substantial case analysis. We hope that there might be some more general reduction of which these are simply special cases, perhaps even a reduction that works for superconstant approximation factors.) It also has the advantage of only making at most two oracle queries.

Theorem 5.1. *There is a deterministic Cook reduction from* $\mathbb{Z}$SVP *to* $\sqrt{2}$-SVP.

Proof. Let $\mathbf{B} \in \mathbb{R}^{n\times n}$ be the input instance of $\mathbb{Z}$SVP with $\mathcal{L} := \mathcal{L}(\mathbf{B})$, and assume without loss of generality that $n \geq 3$.

The reduction works as follows. First, it calls its $\sqrt{2}$-SVP oracle on the input basis $\mathbf{B}$, receiving as output a vector $\boldsymbol{v}$. If $\|\boldsymbol{v}\| = 1$, the reduction returns $\boldsymbol{v}$. Otherwise, it computes a basis $\mathbf{B}'$ of the orthogonal projection $\mathcal{L}' := \pi_{\boldsymbol{v}^\perp}(\mathcal{L})$ of $\mathcal{L}$ onto $\boldsymbol{v}^\perp$, and calls its $\sqrt{2}$-SVP oracle on $\mathbf{B}'$, receiving as output a vector $\boldsymbol{w}$. If $\|\boldsymbol{w}\| = 1$, the reduction returns $\boldsymbol{w}$. Otherwise, it outputs $\boldsymbol{v}/2 + \boldsymbol{w}$.

The reduction is clearly efficient. It remains to analyze its correctness. The algorithm is clearly correct if $\|\boldsymbol{v}\| = 1$, so assume not.

Fix an arbitrary orthogonal basis $\boldsymbol{R} = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n)$ of $\mathcal{L}$ for analysis. Because $\mathcal{L} \cong \mathbb{Z}^n$, if $\boldsymbol{v}$ is not a unit vector then it must be of the form $\pm\boldsymbol{r}_i \pm \boldsymbol{r}_j$ for some $i \neq j$. Indeed, any vector $\boldsymbol{Rx} \in \mathcal{L}$ whose coefficient vector $\boldsymbol{x}$ has a coordinate of magnitude at least 2 will have norm at least 2, and any such vector such that $\boldsymbol{x}$ has at least 3 non-zero coordinates will have norm at least $\sqrt{3}$, so this is the only possibility. It follows that $\mathcal{L}' \cong ((1/\sqrt{2}) \cdot \mathbb{Z}) \oplus \mathbb{Z}^{n-2}$, and therefore that $\lambda_1(\mathcal{L}') = 1/\sqrt{2}$. So, on input $\mathbf{B}'$, a $\sqrt{2}$-SVP oracle will output a vector $\boldsymbol{w}$ of one of two types: (1) a unit-length vector $\boldsymbol{w} = \pm\boldsymbol{r}_k$ for $k \notin \{i, j\}$, or (2) $\boldsymbol{w} = (\pm\boldsymbol{r}_i \pm \boldsymbol{r}_j)/2$ satisfying $v \perp w$. Indeed, these are the only two types of vectors of norm at most $\sqrt{2} \cdot \lambda_1(\pi_{\boldsymbol{v}^\perp}(\mathcal{L})) = 1$. In the former case, we're done since $\boldsymbol{w}$ has unit length and $\boldsymbol{w} \in \mathcal{L}$. In the latter case, because $\boldsymbol{v} = \pm\boldsymbol{r}_i \pm \boldsymbol{r}_j$, $\boldsymbol{w} = (\pm\boldsymbol{r}_i \pm \boldsymbol{r}_j)/2$, and $\boldsymbol{v} \perp \boldsymbol{w}$, we have that $\boldsymbol{v}/2 + \boldsymbol{w}$ is equal to either $\pm\boldsymbol{r}_i$ or $\pm\boldsymbol{r}_j$, all of which are again unit-length vectors in $\mathcal{L}$, as needed. □

5.2 Sparsification

We next present our main algorithm, which works via repeated *random lattice sparsification*, a technique for sampling a random sublattice $\mathcal{L}'$ of the input lattice $\mathcal{L}$ of fixed index originally due to Khot [Kho05]. More specifically, at each iteration our algorithm sparsifies the input lattice $\mathcal{L}$ and then calls its γ-uSVP oracle on the resulting lattice $\mathcal{L}'$. The algorithm succeeds if

$$\lambda_1(\mathcal{L}') = \lambda_1(\mathcal{L}) \text{ and } \gamma \cdot \lambda_1(\mathcal{L}') < \lambda_2(\mathcal{L}') \ , \tag{3}$$

since then the γ-uSVP oracle is guaranteed (with high probability, if the oracle is randomized) to output a shortest non-zero vector in the original lattice $\mathcal{L}$ since $\mathcal{L}' \subseteq \mathcal{L}$

The crux of the analysis of this algorithm is to upper bound the number of times we need to "sparsify and call the γ-uSVP oracle" before the condition in Equation (3) holds. We upper bound this number by roughly A/G, where $G := N_{\text{prim}}(\mathcal{L}, \lambda_1(\mathcal{L}))$ and $A := N_{\text{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L}))$ are (up to a multiplicative factor of 2 arising from N_{prim} counting $\pm\boldsymbol{x}$ as a single vector) the number of primitive vectors of norm at most $\lambda_1(\mathcal{L})$ and norm at most $\gamma \cdot \lambda_1(\mathcal{L})$, respectively. We note in passing that $2G$ is the so-called kissing number of $\mathcal{L}$. (We have used the standard mnemonic of G representing "good" vectors and A representing "annoying" vectors, although here A representing "all" primitive vectors shorter than $\gamma \cdot \lambda_1(\mathcal{L})$, including the good vectors, is more appropriate.)

The following theorem presents our algorithm. Intuitively, it says that exact SVP is not much harder than approximate uSVP on lattices with few short vectors. Namely, it says that there is an algorithm for solving exact SVP in roughly $A/G \cdot T_{\text{uSVP}}(\gamma, n)$ time, where $T_{\text{uSVP}}(\gamma, n)$ denotes the fastest runtime of a (possibly randomized) algorithm for γ-uSVP on lattices of rank n.

Theorem 5.2. *Let $\gamma = \gamma(n) \geq 1$ and let $\mathcal{L}$ be a lattice of rank n. Let $G := N_{prim}(\mathcal{L}, \lambda_1(\mathcal{L}))$ and let $A := N_{prim}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L}))$. Then there is a randomized algorithm that solves (exact)* SVP *on $\mathcal{L}$ in $(A/G) \cdot T_{\text{uSVP}}(\gamma, n) \cdot \text{poly}(n)$ time.*

Proof. It suffices to prove the claim for $\gamma \leq 2^{n/2}$. Indeed, suppose that the claim is true for $\gamma = 2^{n/2}$. Then for any $\gamma > 2^{n/2}$, we can solve SVP on $\mathcal{L}$ in time

$$\begin{aligned} N_{\text{prim}}(\mathcal{L}, 2^{n/2} \cdot \lambda_1(\mathcal{L})) \cdot T_{\text{uSVP}}(2^{n/2}, n) \cdot \text{poly}(n) &\leq N_{\text{prim}}(\mathcal{L}, 2^{n/2} \cdot \lambda_1(\mathcal{L})) \cdot \text{poly}(n) \\ &\leq N_{\text{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L})) \cdot T_{\text{uSVP}}(\gamma, n) \cdot \text{poly}(n) \ , \end{aligned}$$

where the first inequality uses $T_{\text{uSVP}}(2^{n/2}, n) \leq \text{poly}(n)$, which holds by the LLL algorithm [LLL82].

The algorithm for solving exact SVP works as follows. It uses a γ-uSVP oracle, which we assume without loss of generality always outputs a non-zero lattice vector even if the input lattice $\mathcal{L}$ does not satisfy the promise that $\gamma \cdot \lambda_1(\mathcal{L}) < \lambda_2(\mathcal{L})$. Let $\mathbf{B}$ be the input basis of $\mathcal{L}$.

1. Guess A' satisfying $A \leq A' \leq 2A$, and sample a prime p satisfying $60A' \log A' \leq p \leq 120A' \log A'$.
2. Repeat the following $K := \lceil 720A' \log A'/G \rceil \cdot n$ times:
   (a) Sample $\boldsymbol{a} \sim \mathbb{Z}_p^n$ uniformly at random and compute a basis $\mathbf{B}'$ of the sublattice
   $$\mathcal{L}' := \{\boldsymbol{x} \in \mathcal{L} : \langle \mathbf{B}^{-1}\boldsymbol{x}, \boldsymbol{a} \rangle \equiv 0 \pmod{q}\} \ .$$
   (b) Call the γ-uSVP oracle on $\mathbf{B}'$. Let $\boldsymbol{w}$ be the oracle's output.
3. Return a shortest vector among all vectors $\boldsymbol{w}$ found in Step 2b.

We start by proving correctness. Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_A \in \mathcal{L}_{\text{prim}}$ be the unique (up to sign) primitive lattice vectors satisfying $\|\boldsymbol{x}_i\| \leq \gamma r$, $\boldsymbol{x}_i \neq \pm \boldsymbol{x}_j$ for $i \neq j$ ordered so that $\|\boldsymbol{x}_i\| = \lambda_1(\mathcal{L})$ for $i \in [G]$. For a fixed iteration of Step 2, define the events

$$E_i := [\langle \boldsymbol{a}, \mathbf{B}^{-1}\boldsymbol{x}_i \rangle \equiv 0 \pmod{p} \text{ and } \langle \boldsymbol{a}, \mathbf{B}^{-1}\boldsymbol{x}_j \rangle \not\equiv 0 \pmod{p} \ \ \forall j \neq i]$$

for $i = 1, \ldots, A$. We note that if E_i holds for some $i \in [G]$ then the algorithm succeeds. Indeed, then $\mathcal{L}'$ satisfies $\lambda_1(\mathcal{L}') = \lambda_1(\mathcal{L})$ and $\lambda_1(\mathcal{L}') < \lambda_2(\mathcal{L}')/\gamma$ so that $\mathbf{B}'$ is a valid instance of γ-uSVP. In that case, the vector $\boldsymbol{w}$ output in Step 2b will satisfy $\boldsymbol{w} \in \mathcal{L}$, $\|\boldsymbol{w}\| = \lambda_1(\mathcal{L})$, which in turn implies that the vector output in Step 3 will be a shortest non-zero vector of $\mathcal{L}$.

We have that $60A \log A \leq p \leq 120A' \log A' \leq 480A \log A$, and so for all $A \geq 480$,

$$\frac{p}{20 \log p} \geq \frac{60A \log A}{20(\log 480 + \log A + \log \log A)} \geq A \ .$$

(If $A < 480$, then a very similar argument works when p is set to be a sufficiently large fixed prime in Step 1.) Therefore, by the lower bound in Theorem 2.13 (with $\boldsymbol{x}_i$ taking the role of $\boldsymbol{x}_0$ and $A - 1$ taking the role of N there), it holds that

$$\Pr[E_i] \geq \frac{1}{p} - \frac{A}{p^2} \geq \frac{1}{120A' \log A'} - \frac{A}{(60A' \log A')^2} \geq \frac{1}{180A' \log A'} \geq \frac{1}{720A \log A} \ ,$$

where we have used the bounds on A' and p from Step 1. Because the events E_i are disjoint, we further have that the probability of success in a single iteration of Step 2 is at least

$$\Pr\Big[\bigcup_{i \in [G]} E_i\Big] = \sum_{i \in [G]} \Pr[E_i] \geq \frac{G}{720A \log A} \ .$$

The overall probability of success across all K iterations is then at least

$$1 - \Big(1 - \frac{G}{720A \log A}\Big)^K \geq 1 - \exp(-KG/(720A \log A)) \geq 1 - e^{-n} \ .$$

We next turn to upper bounding the algorithm's runtime. In Step 1, guessing A' can be done by setting $A' := 2^\ell$ for integers ℓ satisfying $1 \le \ell \le \log_2(2\gamma+1) \cdot n$. Indeed, this suffices because a straightforward packing argument shows that $A := N_{\mathrm{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L})) \le (2\gamma+1)^n$. Finding p can be done efficiently, say, by repeatedly sampling a uniformly random integer in $[60A' \log A', 120A' \log A']$ and testing whether it is prime. Additionally, the bases $\mathbf{B}'$ computed in Step 2a can be computed efficiently given $\mathbf{B}$ and $\boldsymbol{a}$ (see, e.g., [Ste16a, Claim 2.15]). So, the algorithm's overall runtime is dominated by $T_{\mathrm{uSVP}}(\gamma, n)$ times the product of the number of guesses needed for A' in Step 1 and the number of calls to the γ-uSVP oracle in Step 2b. Because $K = O(A \log A/G)$ and $\log A \le (\log_2(2\gamma+1) \cdot n) \le \mathrm{poly}(n)$, the runtime is therefore at most

$$(\log_2(2\gamma+1) \cdot n) \cdot K \cdot T_{\mathrm{uSVP}}(\gamma, n) \cdot \mathrm{poly}(n) \le (A/G) \cdot T_{\mathrm{uSVP}}(\gamma, n) \cdot \mathrm{poly}(n) \ ,$$

as needed. □

By combining Theorem 5.2 with the point counting bound for $\mathbb{Z}^n$ in Proposition 2.14 we immediately get the following corollary.

Corollary 5.3. *Let $1 \le \gamma \le \sqrt{n/2}$ and let $\mathcal{L}$ be a lattice satisfying $\mathcal{L} \cong \mathbb{Z}^n$. Then there is a randomized algorithm that solves* SVP *on $\mathcal{L}$ in $T_{\mathrm{uSVP}}(\gamma, n) \cdot (20n/\gamma^2)^{\gamma^2} \cdot \mathrm{poly}(n)$ time with probability at least $1 - \exp(-n)$.*

Proof. By the rotational invariance of the ℓ_2 norm and Proposition 2.14,

$$A := N_{\mathrm{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L})) = N_{\mathrm{prim}}(\mathbb{Z}^n, \gamma \cdot \lambda_1(\mathbb{Z}^n)) \le N(\mathbb{Z}^n, \gamma) \le (20n/\gamma^2)^{\gamma^2} \ .$$

The result then follows immediately from Theorem 5.2. □

Combining Corollary 5.3 with the fast algorithm for (large) constant factor approximate SVP from Theorem 2.9 leads to a roughly $2^{0.802n}$-time algorithm for $\mathbb{Z}$SVP. We again emphasize that this substantially improves over the roughly 2^n-time SVP algorithm for general lattices from [ADRS15], which was also the previous fastest known algorithm for $\mathbb{Z}$SVP.

Corollary 5.4. *Then for every constant $\varepsilon > 0$ there is a randomized algorithm that solves* SVP *on lattices $\mathcal{L}$ satisfying $\mathcal{L} \cong \mathbb{Z}^n$ in $2^{(0.802+\varepsilon)n} \cdot \mathrm{poly}(n)$ time with probability at least $1 - \exp(-n)$.*

Proof. The result follows from Corollary 5.3 together with the upper bound on T_{uSVP} in Equation (2) and the fact that $(20n/\gamma^2)^{\gamma^2} \le \mathrm{poly}(n)$ for constant γ. □

As mentioned, one may interpret the algorithm in Theorem 5.2 as a Turing reduction from $\mathbb{Z}$SVP to γ-uSVP. Using this interpretation leads to a new *hardness* result for approximate uSVP. Namely, if one assumes that there is no randomized polynomial time algorithm for $\mathbb{Z}$SVP then there is also no randomized polynomial time algorithm for solving γ-uSVP for any constant $\gamma \ge 1$. This is notable since γ-uSVP is not known to be NP-hard (or to the best of our knowledge, known to be hard under any other generic complexity-theoretic assumption) for any constant $\gamma > 1$. Indeed, it is only known to be NP-hard (under randomized reductions) for $\gamma = 1 + 1/\mathrm{poly}(n)$; see [AD16, Ste16b]. We get similar hardness for the α-Bounded Distance Decoding Problem (α-BDD), the problem in which, given a (basis of a) lattice $\mathcal{L}$ and a target point $\boldsymbol{t}$ satisfying $\mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \le \alpha \cdot \lambda_1(\mathcal{L})$ as input, the goal is to output a closest lattice point to $\boldsymbol{t}$ (i.e., $\boldsymbol{x} \in \mathcal{L}$ satisfying $\|\boldsymbol{t} - \boldsymbol{x}\| = \mathrm{dist}(\boldsymbol{t}, \mathcal{L})$). Indeed, this additional hardness for BDD follows almost immediately from a known reduction from uSVP to BDD [LM09].

Corollary 5.5. *Suppose that there is no randomized* $\mathrm{poly}(n)$*-time algorithm* $\mathbb{Z}$SVP. *Then there is no randomized* $\mathrm{poly}(n)$*-time algorithm for γ-*uSVP *for any constant $\gamma \ge 1$ or for α-*BDD *for any constant $\alpha > 0$.*

Proof. The contrapositive of the claim for uSVP follows immediately from Corollary 5.3. The claim for BDD follows from this by additionally noting that [LM09] gives an efficient reduction from γ-uSVP to $(1/\gamma)$-BDD for any $\gamma = \gamma(n) \le \mathrm{poly}(n)$, and that α'-BDD trivially reduces to α-BDD for $\alpha' \le \alpha$. □

We also note that our main reduction is related to an interesting question of Ducas and van Woerden [DvW21a], which asks whether there is a reduction from exact SVP on "f-unusual" lattices—essentially lattices for which Minkowski's Theorem (or, more-or-less equivalently, the Gaussian heuristic) is loose by a factor of at least f—to (approximate) uSVP. Rotations of $\mathbb{Z}^n$ are $\Theta(\sqrt{n})$-unusual in this sense, and so our reduction can be viewed as answering a special case of this question. (The reduction does not work from general f-unusual-SVP.)

Exponential-time variants of the corollaries above. Finally, we remark that there are interesting "exponential-time variants" of the algorithmic result in Corollary 5.4 and the hardness result in Corollary 5.5. These follow from analyzing $N(\mathbb{Z}^n, \gamma)$ with $\gamma = \gamma(n) \approx \sqrt{n}$. Namely, by Proposition 2.14 it holds that for every constant $c > 0$ there exists a constant $c' > 0$ such that $N(\mathbb{Z}^n, c'\sqrt{n}) \leq 2^{cn}$.

Indeed, this bound on $N(\mathbb{Z}^n, c'\sqrt{n})$ together with Corollary 5.3 implies that if $T_{\text{uSVP}}(\gamma, n) = 2^{cn} \cdot \text{poly}(n)$ holds for some $\gamma = o(\sqrt{n})$ and constant $c > 0$, then for every constant $\varepsilon > 0$ there exists a $(2^{(c+\varepsilon)n} \cdot \text{poly}(n))$-time algorithm for $\mathbb{Z}$SVP. I.e., any exponential-time algorithm for γ-uSVP with $\gamma = o(\sqrt{n})$ can be turned into an algorithm with a similar runtime for $\mathbb{Z}$SVP. On the other hand, this bound together with Corollary 5.3 implies that if there is no $2^{o(n)}$-time algorithm for $\mathbb{Z}$SVP then there is no $2^{o(n)}$-time algorithm for γ-uSVP with $\gamma = o(\sqrt{n})$. I.e., exponential hardness of $\mathbb{Z}$SVP implies exponential hardness of γ-uSVP with $\gamma = o(\sqrt{n})$.

6 Experiments

The code and raw data for our experiments can be found at [BGPS21].

6.1 Experiments on different procedures for generating bases

In this section, we present experimental results examining the effectiveness of standard basis reduction algorithms for solving $\mathbb{Z}$SVP. Specifically, we generate bases of $\mathbb{Z}^n$ (which we then treat as instances of $\mathbb{Z}$SVP) using three procedures: discrete-Gaussian-based sampling, unimodular-matrix-product-based sampling, and Bézout-coefficient-based sampling. Using each of these procedures, we generate bases in dimensions $n = 128$, 256, and 512 with a variety of settings for procedure-specific parameters.[7] These results extend those in [BM21], which included experiments on bases generated using the second two procedures.

For each basis generating procedure (and corresponding set of parameters), we run the LLL algorithm and BKZ reduction algorithm (as implemented in fplll [FPL]) with different block sizes. For BKZ, we use block sizes 3, 4, 5, 10, and 20—though in dimension 512, we left out block size 20 for most of our experiments due to computational constraints. We run these algorithms sequentially. That is, we run BKZ with block size 3 on the matrix returned by the LLL algorithm, we run BKZ with block size 4 on the matrix returned by BKZ with block size 3, and so forth.

For each parameter set of each basis generation procedure, we performed this experiment twenty times (except in a few cases in which we only performed nineteen), and we report below on the smallest block size that found a shortest non-zero vector in the lattice (where we think of LLL as BKZ with block size 2), if one was found. More data, such as the running times of our experiments, the squared length of the shortest vector found in each trial, and the code used, can be found in the associated repository [BGPS21].

At a high level, the data tell a relatively simple story. We were able to find a shortest vector in all cases in dimension 128 (often with block size 10). In dimensions 256 and 512, we were generally unable to find shortest vectors when the basis was generated with "reasonable parameters," where the definition of which parameters settings are reasonable of course depends on the procedure used to generate the basis.

[7]We note that these experiments were actually performed on bases of $\mathbb{Z}^n$ itself—not rotations of $\mathbb{Z}^n$—because this allows us to work with bases with integer entries. Of course, it is not actually hard to find a short vector in this case—simply ignore the input and output $\boldsymbol{e}_1$. However, the experiments that we perform use algorithms that are rotation-invariant—that is, their performance is unchanged if we apply a rotation to the input basis. The results of our experiments would therefore be essentially identical if we had run them on rotations of $\mathbb{Z}^n$, though the experiments were performed on $\mathbb{Z}^n$ itself.

We note that the "reasonable parameter" regime for the discrete Gausian-based generating procedure yields significantly shorter vectors than the other procedures, which might be considered an advantage, as it allows for more efficient representations.

6.1.1 Discrete Gaussian-based sampling

We start by presenting the results of experiments performed on bases generated essentially as described in Section 3 (which is also what we use for our encryption scheme in Section 4). However, we make three minor modifications. First, instead of sampling vectors one at a time until we find a generating set of $\mathbb{Z}^n$, we simply sample $n + 10$ vectors. (There is nothing special about the number 10 here.) Empirically, we found that this yielded a generating set with high probability. Notice that this is much better than what is proven in Lemma 2.5. See also [NP21].

Second, recall that the basis sampling procedure in Section 3 requires an algorithm $\mathcal{A}$ that converts such a generating set into a basis (and is rotation invariant), as does our description of the sampling technique below. Since LLL is such an algorithm, and since we intend to run LLL anyway, we simply skip this step and run LLL directly on the generating set. Third, we do not bother to apply a rotation to the basis, because the algorithms that we are running are invariant under rotation (as noted in Footnote 7).

Basis Generating Procedure 1: Discrete Gaussian-based sampling

Input: A dimension n, a parameter $s > 0$.

Algorithm:

1. For all $1 \leq i \leq n$ and $1 \leq j \leq n + 10$ sample $m_{i,j}$ from a discrete Gaussian with parameter s and set $\mathbf{M} \leftarrow (m_{i,j})$.
2. Apply an algorithm $\mathcal{A}$ that converts a generating set into a basis $\mathbf{B}$, as in Definition 3.1.
3. If $\det(\mathbf{B}) \neq \pm 1$, start over.

In our experiments, we took $s \in \{1, 10, 1000\}$. See Table 1. Setting $s = 1$ is not a "reasonable" parameter choice, as the resulting vectors are quite sparse. (Each coordinate of each vector in the generating set is zero with probability roughly 0.92.) In particular, we would certainly *not* recommend using parameter $s = 1$ for cryptography, and we include data with $s = 1$ only for completeness. Nevertheless, interestingly, in all twenty runs, we were actually unable to find a shortest vector even for $s = 1$ in dimension $n = 512$.

For $s = 10$ and $s = 1000$, we found shortest vectors in dimension $n = 128$ (as we did in all experiments in $n = 128$ dimensions) and failed to find shortest vectors in dimensions $n = 256$ and $n = 512$. The data suggest that there was not too much difference between parameter $s = 10$ and parameter $s = 1000$. E.g., in dimension $n = 128$, there is no obvious difference between the block size needed to break the $s = 10$ case and the block size needed to break the $s = 1000$ case. (In contrast, LLL was able to break the $s = 1$ case.)

6.1.2 Unimodular matrix product

The second basis sampling technique that we analyze was proposed in [BM21], where it is called Algorithm 3.[8] To introduce it, we start by discussing a family of embedding maps $\phi_{k_1,\ldots,k_d} : \mathbb{R}^{d \times d} \to \mathbb{R}^{n \times n}$ for size d subsets of indices $\{k_1, \ldots, k_d\} \subseteq \{1, \ldots, n\}$ that embed a smaller $d \times d$ matrix H into a larger $n \times n$ matrix $\phi(H)$:

$$(\phi_{k_1,\ldots,k_d}(H))_{i',j'} = \begin{cases} H_{i,j} & \text{if } i' = k_i \text{ and } j' = k_j \text{ for some } i, j \leq d; \\ 1_{i'=j'} & \text{otherwise,} \end{cases}$$

[8] Algorithm 1 in [BM21] is not a polynomial-time algorithm, and is not intended to be used. Algorithm 2 is essentially (though not quite) the special case of Algorithm 3 with $d = 2$. This is why we do not consider Algorithms 1 and 2 in this work.

where $H = (H_{i,j}) \in \mathbb{R}^{d\times d}$ and $\phi_{k_1,\ldots,k_d}(H) = H' = (H'_{i',j'}) \in \mathbb{R}^{n\times n}$. With this, we can define the next basis sampling technique:

Basis Generating Procedure 2: Unimodular matrix product

Input: A dimension n, a block size $2 \le d \le n$, a size bound $B \ge 1$, a word length $L \ge 1$.

Algorithm:

```
1. Set a matrix $\mathbf{A} \leftarrow I_n$.
2. For all $1 \le i \le d$ and $1 \le j \le d$ sample the integer $m_{i,j}$ uniformly at random from $[-B, B]$.
3. Set $\mathbf{M} \leftarrow (m_{i,j})$.
4. If $\det(\mathbf{M}) \neq \pm 1$:
    • Then repeat steps two and three.
    • Otherwise sample distinct integers $k_1, \ldots, k_d$ uniformly at random from $[1, n]$, then set
      $\mathbf{M}' \leftarrow \phi_{k_1,\ldots,k_d}(\mathbf{M})$ and reassign $\mathbf{A} \leftarrow \mathbf{A}\mathbf{M}'$
5. Repeat steps two through four a total of $L$ times then output $\mathbf{A}$.
```

In our experiments, we considered all combinations of parameters $d \in \{2, 3, 4\}$, $B = 1$, and $L \in \{10n, 20n, 30n, 40n, 50n\}$, except that we did perform experiments with some of the larger parameter choices when $n = 512$ when our experiments failed to find short vectors with smaller parameters. See Table 2. (These parameter settings are roughly in line with those studied in in [BM21].)

As in all of our experiments, we were able to find a shortest non-zero vector for all choices of parameters in dimension 128. For all but the smallest parameter settings, the required block size to do so was broadly comparable to what we saw for the Gaussian—typically block size 10 was required, but occasionally smaller block sizes sufficed.

We include the case of $d = 2$ and $L = 10n$ for completeness, but we note that this is a rather silly example, as bases generated in this way often contain quite short vectors. Indeed, the expected number of vectors in the resulting basis that have length exactly one is at least $n(1 - 4/(5n))^L \approx n/3000$, and indeed the data show that the generated basis often itself contains a shortest non-zero vector in this case. (Asymptotically, one should clearly take $L \ge \lambda n$, where λ is the security parameter.)

More generally, the data make clear that the success of our experiments depended heavily on the parameters. E.g., even in dimension 512, the LLL algorithm was able to break the $d = 2$ and $L = 20n$ case, but none of our experiments broke larger parameter settings.

A strange anomaly. We note that our data differ from the data in [BM21] in a surprising way. Specifically, in dimension $n = 500$, with parameters $d = 2$, $B = 1$, and $L = 30n = 15000$, [BM21] successfully recovered a shortest non-zero vector in their (single-trial) experiment. In dimension $n = 512$ with parameters $d = 2$, $B = 1$, and $L = 30n = 15360$, we failed to find a shortest non-zero vector in all twenty trials of our experiment. Similarly, for $d = 3$, $B = 1$, and $L = 10n$, we failed to find a shortest non-zero vector in all twenty trials of our experiment, while [BM21] did find a shortest non-zero vector. This is in spite of the fact that we run BKZ with block size 10 (in addition to block sizes 2, 3, 4, and 5), while [BM21] only uses block sizes 2, 3, and 4.

One might argue that this is due to the slightly different choice of dimension—we performed the relevant experiments in dimension $n = 512$, while [BM21] performed them in dimension 500. However, we ran the first experiment (i.e., $d = 2$, $B = 1$, and $L = 30n$) again with $n = 500$ and found the same result. So, this does not offer an explanation.

We suspect that the most likely explanation is that the specific implementation of basis reduction used in [BM21] happens to perform significantly better than the implementation we used. Specifically, [BM21] used Magma's basis reduction algorithms, while we used fplll.

We do not, however, feel that this anomaly changes the high-level message of these experiments.

6.1.3 Bézout-coefficient-based construction

Given the matrix $\mathbf{M} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n-1}) \in \mathbb{R}^{n\times(n-1)}$, if (and only if) all the minors in $\mathbf{M}$ of size $n-1$ have no non-trivial common factor, then there exists a vector $\boldsymbol{a}$ for which the matrix $\boldsymbol{M}' := (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n-1}, \boldsymbol{a})$ is unimodular. Moreover, if this is the case, then we can find such a vector $\boldsymbol{a}$ efficiently using the extended Euclidean algorithm.

This procedure was suggested by Joseph Silverman and studied in [BM21]. Indeed, [BM21] noted that the resulting vector $\boldsymbol{m}_n$ will typically be *quite* large, and for this reason, they performed a least-squares-based procedure to find an integer linear-combination $\boldsymbol{y}$ of $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n-1}$ such that $\boldsymbol{m}_n' := \boldsymbol{m}_n - \boldsymbol{y}$ is not as large. They then output the basis with $\boldsymbol{m}_n'$ instead of $\boldsymbol{m}_n$. We do not bother with this step, as we note that the LLL algorithm will behave identically on the basis with $\boldsymbol{m}_n$ and the basis with $\boldsymbol{m}_n'$. Indeed, because our experiments start by running the LLL algorithm on $\boldsymbol{M}'$, this step is unnecessary for our experiments.

Basis Generating Technique 3: Bézout-coefficient-based construction

Input: A dimension n, a size bound $B \geq 1$.

Algorithm:

```
1. For all $1 \leq i \leq n$ and $1 \leq j \leq n-1$ sample the integer $m_{i,j}$ uniformly at random from $[-B, B]$.
2. Set $\mathbf{M} := (m_{i,j}) = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n-1})$, and for $1 \leq k \leq n$ let $\mathbf{M}_k$ refer to the submatrix of $\mathbf{M}$
   obtained by removing its $k$-th row.
3. If $\gcd(\det(\mathbf{M}_1), \ldots, \det(\mathbf{M}_n)) \neq 1$:
     • Then repeat steps one and two.
     • Otherwise run the Euclidean Algorithm to find the Bézout coefficients $a_1, \ldots, a_n$ such that
       $\sum_{k=1}^{n} a_k \cdot \det(\mathbf{M}_k) = \pm 1$ (where the sign of the sum is chosen uniformly at random).
4. Output $\boldsymbol{M}' := (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n-1}, \boldsymbol{a}) \in \mathrm{GL}_n(\mathbb{Z})$, where $\boldsymbol{a}^T = (a_1, \ldots, a_n)$.
```

In our experiments, we took $B \in \{1, 10, 100\}$. See Table 3.

We found that the effect of the parameter B was not discernible in our experiments. Indeed, for dimensions 256 and 512, our algorithms failed to find a shortest vector for all choices of B, including $B = 1$. And, in dimension 128, we found a shortest vector in all cases (as we always did), but the block size needed shows no obvious dependence on B. These results are quite similar to those in [BM21].

6.2 An interesting threshold phenomenon

In our data, we noticed a curious phenomenon. We found that the shortest vector in the bases returned by our algorithms almost always had either length one or had length larger than roughly $\sqrt{n}/2$. (We provide much more detail below.) The algorithms rarely found vectors in between. We refer to this as a *threshold phenomenon*, i.e., there is some *threshold* length $r \approx \sqrt{n}/2$ such that, if a basis reduction algorithm finds a vector with length less than r, then it typically will find the shortest vector.

To determine whether this was a true phenomenon, we conducted 1000 additional experiments with bases generated as described in Section 6.1.1 in dimension $n = 128$ and with parameter $s = 1000$. In order to get

		block size						
n	s	2	3	4	5	10	20	unbroken
128	1	20	0	0	0	0	0	0
128	10	0	0	1	1	18	0	0
128	1000	0	0	0	3	17	0	0
256	1	2	2	1	0	3	3	9
256	10	0	0	0	0	0	0	20
256	1000	0	0	0	0	0	0	20
512	1	0	0	0	0	0	0	20
512	10	0	0	0	0	0	0	20
512	1000	0	0	0	0	0	0	20

Table 1: Experimental results for basis reduction performed on bases generated using the discrete-Gaussian-based construction described in Section 6.1.1. The entries under each block size represent the number of times (out of a total of twenty experiments) that a shortest non-zero vector was found with a given block size (but no smaller block size), and the entries in the "unbroken" column represent the number of times that we failed to find a shortest non-zero vector. Non-zero entries are highlighted.

more fine-grained results than those from Section 6.1, we ran LLL and then sequentially ran BKZ with all block sizes between 3 and 10. For these experiments, we were not particularly interested in the block size that was needed to find a shortest non-zero vector.[9] Instead, we were interested in the length of the shortest vector found by a run of BKZ when it did not find a vector with length one.

The results are striking. See Figure 1 and Figure 2. In particular, in 1000 experiments, our algorithms never returned a basis whose shortest vector had length between $\sqrt{3}$ and $\sqrt{28}$. (Strangely, there are four examples when the shortest vector in the basis had length $\sqrt{2}$. We have no explanation for this.) And, e.g., there are only fifteen examples in which this length was less than $\sqrt{35}$.

Similar phenomena are well documented and well understood in the case of unique SVP, or more generally when the lattice has remarkably short vectors which together span a lower-dimensional sublattice (e.g., in NTRU lattices) [KF17, AGVW17, DvW21b, AD21]. In both of these cases, one can argue that BKZ should find a short vector once it finds a lower-dimensional sublattice that is guaranteed to contain any short vectors. For $\mathbb{Z}^n$, this phenomenon seems at least a little bit different, as $\mathbb{Z}^n$ is spanned by its shortest vectors—so there is no strict sublattice containing all of them. However, Ducas and van Woerden predict [DvW21a] that a phenomenon like this should occur for all "unusual" lattices—that is, lattices for which $\lambda_1(\mathcal{L}) \ll \sqrt{n}\det(\mathcal{L})^{1/n}$— of which $\mathbb{Z}^n$ is an example, with a precise quantitative prediction that seems to match our experiments.

One might also draw an analogy here with our reduction in Section 5.1, which showed how projecting orthogonally to a vector with length $\sqrt{2}$ can aid in finding a shortest non-zero vector. Indeed, as we mention there, we know (rather complicated) generalizations that work for $\sqrt{3}$ and $2 = \sqrt{4}$, and we suspect that one can generalize this much further. Perhaps one can view the BKZ algorithm as doing something similar? One might also try to draw an analogy here with Szydlo's heuristic algorithm [Szy03].

[9] We found a shortest non-zero vector eventually in all experiments, and only 8 of the 1000 trials required block size 10 to find a shortest non-zero vector. See [BGPS21] for the raw data.

				block size						
n	B	L	d	2	3	4	5	10	20	unbroken
128	1	1280	2	20	0	0	0	0	0	0
128	1	2560	2	0	0	1	3	16	0	0
128	1	3840	2	0	0	1	5	14	0	0
128	1	5120	2	0	0	1	3	16	0	0
128	1	6400	2	0	0	0	2	18	0	0
128	1	1280	3	0	0	2	5	13	0	0
128	1	2560	3	0	0	0	4	16	0	0
128	1	3840	3	0	0	1	5	14	0	0
128	1	5120	3	0	0	1	4	15	0	0
128	1	6400	3	0	0	1	4	15	0	0
128	1	1280	4	0	0	1	5	14	0	0
128	1	2560	4	0	0	3	5	12	0	0
128	1	3840	4	0	0	2	4	14	0	0
128	1	5120	4	0	1	3	2	14	0	0
128	1	6400	4	0	0	0	4	16	0	0
256	1	2560	2	20	0	0	0	0	0	0
256	1	5120	2	0	0	0	0	0	0	20
256	1	7680	2	0	0	0	0	0	0	20
256	1	10240	2	0	0	0	0	0	0	20
256	1	12800	2	0	0	0	0	0	0	20
256	1	2560	3	0	0	0	0	0	0	20
256	1	5120	3	0	0	0	0	0	0	20
256	1	7680	3	0	0	0	0	0	0	20
256	1	10240	3	0	0	0	0	0	0	20
256	1	12800	3	0	0	0	0	0	0	20
256	1	2560	4	0	0	0	0	0	0	20
256	1	5120	4	0	0	0	0	0	0	20
256	1	7680	4	0	0	0	0	0	0	20
256	1	10240	4	0	0	0	0	0	0	20
256	1	12800	4	0	0	0	0	0	0	20
512	1	5120	2	20	0	0	0	0	▨	0
512	1	10240	2	20	0	0	0	0	▨	0
512	1	15360	2	0	0	0	0	0	▨	20
512	1	20480	2	0	0	0	0	0	▨	20
512	1	25600	2	0	0	0	0	0	▨	20
512	1	5120	3	0	0	0	0	0	▨	20
512	1	10240	3	0	0	0	0	0	▨	19
512	1	15360	3	0	0	0	0	0	▨	19
512	1	5120	4	0	0	0	0	0	▨	19

Table 2: Experimental results for basis reduction performed on bases generated using the product of sparse unimodular matrices method described in Section 6.1.2. The entries under each block size represent the number of times (out of a total of twenty trials—except for the last four rows, where we did nineteen trials) that a shortest non-zero vector was found with a given block size (but no smaller block size), and the entries in the "unbroken" column represent the number of times that we failed to find a shortest non-zero vector. Non-zero entries are highlighted. Cells that are grayed out represent block sizes that were not tested.

		block size						
n	B	2	3	4	5	10	20	unbroken
128	1	0	0	0	3	17	0	0
128	10	0	0	1	2	17	0	0
128	100	0	0	1	6	13	0	0
256	1	0	0	0	0	0	0	20
256	10	0	0	0	0	0	0	20
256	100	0	0	0	0	0	0	20
512	1	0	0	0	0	0	▨	20
512	10	0	0	0	0	0	▨	20
512	100	0	0	0	0	0	▨	20

Table 3: Experimental results for basis reduction performed on bases generated using the Bézout-coefficient-based construction described in Section 6.1.3. The entries under each block size represent the number of times (out of a total of twenty experiments) that a shortest non-zero vector was found with a given block size (but no smaller block size), and the entries in the "unbroken" column represent the number of times that we failed to find a shortest non-zero vector. Non-zero entries are highlighted. Cells that are grayed out represent block sizes that were not tested.

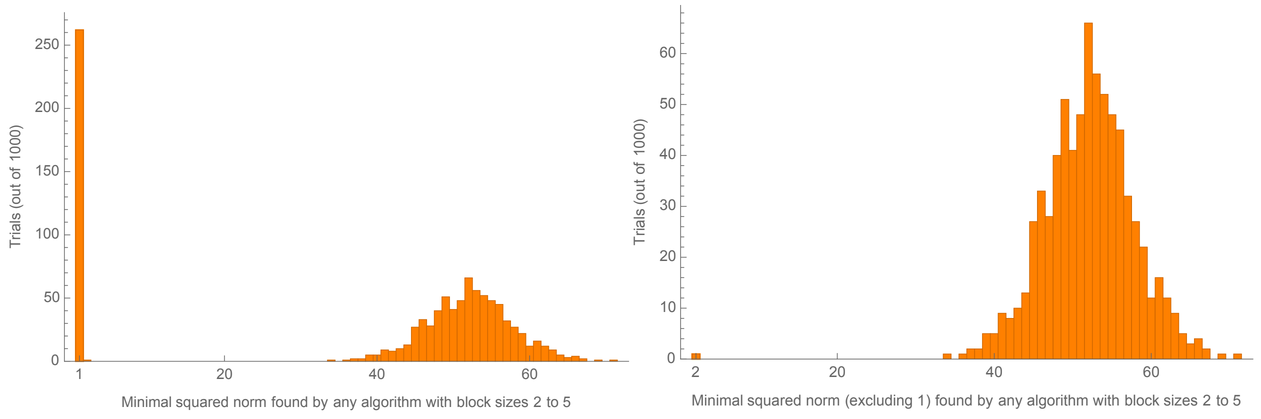

Figure 1: On the left is a histogram of the squared norm of the shortest vector found by BKZ with block size ≤ 5 for discrete Gaussian bases with $n = 128$ and $s = 1000$. On the right is the same histogram without the trials where this norm was 1.

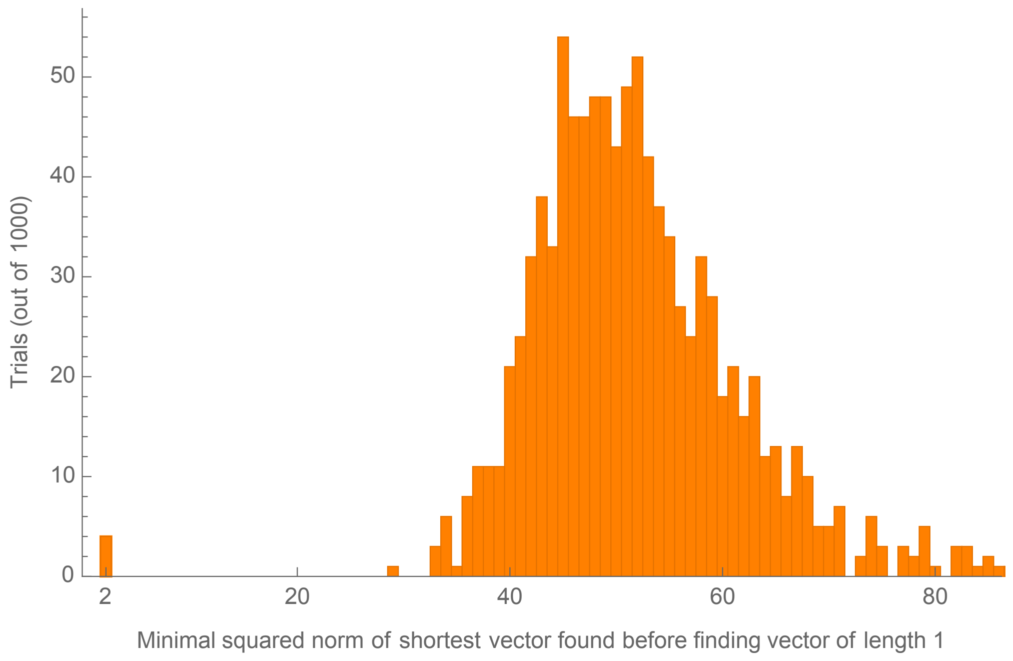

Figure 2: A histogram of the minimal squared length of a basis vector found before finding a vector with length one after 1000 trials on discrete Gaussian bases with $n = 128$ and $s = 1000$. E.g., if in one experiment BKZ with block size 6 finds a vector with length one, but the shortest vector found by BKZ with block sizes 2 through 5 had squared norm 40, then this experiment would count as 40.

References

[AD16] Divesh Aggarwal and Chandan K. Dubey. Improved hardness results for unique shortest vector problem. *Inf. Process. Lett.*, 116(10):631–637, 2016. 2, 18

[AD21] Martin Albrecht and Léo Ducas. Lattice attacks on NTRU and LWE: A history of refinements, 2021. 23

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete Gaussian sampling: Extended abstract. In *STOC*, 2015. 1, 8, 15, 18

[AGVW17] Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In *ASIACRYPT*, 2017. 23

[AUV19] Divesh Aggarwal, Bogdan Ursu, and Serge Vaudenay. Faster sieving algorithm for approximate SVP with constant approximation factors. 2019. 2, 8

[BGPS21] Huck Bennett, Atul Ganju, Pura Peetathawatchai, and Noah Stephens-Davidowitz. Experiments on solving SVP on rotations of $\mathbb{Z}^n$. `https://github.com/poonpura/Experiments-on-Solving-SVP-on-Rotations-of-Z-n`, 2021. 19, 23

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of Learning with Errors. In *STOC*, 2013. 7

[BM21] Tamar Lichter Blanks and Stephen D. Miller. Generating cryptographically-strong random lattice bases and recognizing rotations of $\mathbb{Z}^n$. In *PQCrypto*, 2021. i, 4, 5, 19, 20, 21, 22

[BSW16] Shi Bai, Damien Stehlé, and Weiqiang Wen. Improved reduction from the bounded distance decoding problem to the unique shortest vector problem in lattices. In *ICALP*, 2016. 2

[CDLP13] Kai-Min Chung, Daniel Dadush, Feng-Hao Liu, and Chris Peikert. On the lattice smoothing parameter problem. In *CCC*, 2013. 3

[CGG17] Karthekeyan Chandrasekaran, Venkata Gandikota, and Elena Grigorescu. Deciding orthogonality in Construction-A lattices. *SIAM Journal on Discrete Mathematics*, 31(2):1244–1262, January 2017. 1, 4

[DR16] Daniel Dadush and Oded Regev. Towards strong reverse Minkowski-type inequalities for lattices. In *FOCS*, 2016. 3, 14

[DvW21a] Léo Ducas and Wessel van Woerden. On the Lattice Isomorphism Problem, quadratic forms, remarkable lattices, and cryptography, 2021. `https://eprint.iacr.org/2021/1332`. i, 3, 4, 5, 10, 19, 23

[DvW21b] Léo Ducas and Wessel van Woerden. Ntru fatigue: How stretched is overstretched? In *ASIACRYPT*, 2021. 23

[ERS21] Yael Eisenberg, Oded Regev, and Noah Stephens-Davidowitz. A tight reverse Minkowski inequality for the Epstein zeta function. 2021. 3, 14

[EV20] Friedrich Eisenbrand and Moritz Venzin. Approximate CVP$_p$ in time $2^{0.802n}$. In *ESA*, 2020. 8

[FPL] FPLLL development team. fplll, a lattice reduction library, Version: 5.4.1. Available at `https://github.com/fplll/fplll`. 19

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008. 3, 7

[GS02] Craig Gentry and Michael Szydlo. Cryptanalysis of the revised NTRU signature scheme. In *EUROCRYPT*, 2002. 1, 4

[GS03] Katharina Geißler and Nigel P. Smart. Computing the $M = U^T U$ integer matrix decomposition. In *Cryptography and Coding*, 2003. 1, 4

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963. 9

[HR14] Ishay Haviv and Oded Regev. On the Lattice Isomorphism Problem. In *SODA*, 2014. 3, 5, 7, 10

[Hun19] Christoph Hunkenschröder. Deciding whether a lattice has an orthonormal basis is in co-NP, 2019. 1

[KF17] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In *EUROCRYPT*, 2017. 23

[Kho05] Subhash Khot. Hardness of approximating the Shortest Vector Problem in lattices. *Journal of the ACM*, 52(5):789–808, September 2005. 2, 16

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982. 17

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem. In *CRYPTO*, 2009. 2, 18

[LS14] H. W. Lenstra and A. Silverberg. Revisiting the Gentry-Szydlo algorithm. In *CRYPTO*, 2014. 1, 4

[LS17] H. W. Lenstra and A. Silverberg. Lattices with symmetry. *Journal of Cryptology*, 30(3):760–804, 2017. 1, 4

[LWXZ11] Mingjie Liu, Xiaoyun Wang, Guangwu Xu, and Xuexin Zheng. Shortest lattice vectors in the presence of gaps. *IACR Cryptol. ePrint Arch.*, 2011. 2, 8

[MH73] John Willard Milnor and Dale Husemöller. *Symmetric Bilinear Forms.* Springer-Verlag, 1973. 15

[MO90] J. E. Mazo and A. M. Odlyzko. Lattice points in high-dimensional spheres. *Monatshefte für Mathematik*, 110:47–61, March 1990. 9

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal of Computing*, 37(1):267–302, 2007. 7

[NP21] Phong Q. Nguyen and Löic Pujet. The probability of primitive sets and generators in lattices, 2021. 6, 20

[Pei16] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016. 1

[RS17] Oded Regev and Noah Stephens-Davidowitz. A reverse Minkowski theorem. In *STOC*, 2017. 3, 14

[RS21] Oded Regev and Noah Stephens-Davidowitz. A reverse Minkowski theorem for integral lattices. 2021. 3, 14

[Ste16a] Noah Stephens-Davidowitz. Discrete gaussian sampling reduces to CVP and SVP. In *SODA*. SIAM, 2016. 9, 18

[Ste16b] Noah Stephens-Davidowitz. Search-to-decision reductions for lattice problems with approximation factors (slightly) greater than one. In *APPROX*, 2016. 2, 18

[Ste17] Noah Stephens-Davidowitz. *On the Gaussian measure over lattices.* Ph.D. thesis, New York University, 2017. 9

[Ste20] Noah Stephens-Davidowitz. Lattice algorithms. `https://www.youtube.com/watch?v=o4Pl-0Q5-q0`, 2020. Talk as part of the Simons Institute's semester on lattices. 2, 15

[Szy03] Michael Szydlo. Hypercubic lattice reduction and analysis of GGH and NTRU signatures. In *EUROCRYPT*, 2003. 1, 4, 23

[WLW15] Wei Wei, Mingjie Liu, and Xiaoyun Wang. Finding shortest lattice vectors in the presence of gaps. In *CT-RSA*, 2015. 2, 8

A Proof of Lemma 2.1

We first study the expectation of $(X \bmod \mathbb{Z})^2$ for $X \sim D_s$. We then use the Chernoff-Hoeffding (Lemma 2.15) bound to obtain the result. For $t \in (-1/2, 1/2]$, let $p(t) := (1/s) \cdot \sum_{z\in\mathbb{Z}} \rho_s(t+z)$ be the probability density function of $X \bmod \mathbb{Z} \in (-1/2, 1/2]$. By the Poisson summation formula, we have

$$p(t) = \sum_{w\in\mathbb{Z}} \rho_{1/s}(w)\cos(2\pi wt) = 1 + 2\cos(2\pi t)\exp(-\pi s^2) + 2\sum_{w=2}^{\infty} \cos(2\pi wt)\rho_{1/s}(w) .$$

We write $f(t) := 2\sum_{w=2}^{\infty}\cos(2\pi wt)\rho_{1/s}(w)$ It follows that

$$\mathop{\mathbb{E}}_{X\sim D_s}[(X \bmod \mathbb{Z})^2] = \int_{-1/2}^{1/2} t^2 p(t)\,\mathrm{d}t = 1/12 - \frac{\exp(-\pi s^2)}{2\pi^2} + \int_{-1/2}^{1/2} t^2 f(t)\,\mathrm{d}t\ .$$

Notice that

$$|f(t)| < 2\exp(-4\pi s^2) + 2\int_2^{\infty}\rho_{1/s}(w)\,\mathrm{d}w \le 2\exp(-4\pi s^2) + \int_2^{\infty} w\rho_{1/s}(w)\,\mathrm{d}w = \exp(-4\pi s^2)(2+1/(2\pi s^2))\ .$$

Therefore,

$$\mu := \mathop{\mathbb{E}}_{X\sim D_s}[(X \bmod \mathbb{Z}^2)] = \frac{1}{12} - \frac{\exp(-\pi s^2)}{2\pi^2} + \chi\ ,$$

for some χ with

$$|\chi| \le \exp(-4\pi s^2)\cdot(2+1/(2\pi s^2))\int_{-1/2}^{1/2} t^2\,\mathrm{d}t = \frac{\exp(-4\pi s^2)}{12}\cdot(2+1/(2\pi s^2)) \le \varepsilon_0\ .$$

By the Chernoff-Hoeffding bound (Lemma 2.15),

$$\Pr_{\boldsymbol{X}\sim D_s^n}[|\mathrm{dist}(\boldsymbol{X},\mathbb{Z}^n)^2 - \mu n| \ge \delta n] = \Pr\Big[\Big|\sum(X_i \bmod \mathbb{Z})^2 - \mu n\Big| \ge \delta n\Big] \le 2\exp(-\delta^2 n/10)\ .$$

It follows that

$$\Pr_{\boldsymbol{X}\sim D_s^n}[|\mathrm{dist}(\boldsymbol{X},\mathbb{Z}^n)^2 - \nu| \ge \varepsilon n] \le \Pr_{\boldsymbol{X}\sim D_s^n}[|\mathrm{dist}(\boldsymbol{X},\mathbb{Z}^n)^2 - \mu n| \ge (\varepsilon-\varepsilon_0)n] \le 2\exp(-(\varepsilon-\varepsilon_0)^2 n/10)\ ,$$

as needed.