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

Huck Bennett[2], Atul Ganju[1], Pura Peetathawatchai[3], and Noah Stephens-Davidowitz[1]

[1] Cornell University
[2] Oregon State University
[3] Stanford University

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 very 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, if $\mathbb{Z}$SVP actually *is* hard, then what consequences would follow? 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 $2^{n/2+o(n)}$-time algorithm for $\mathbb{Z}$SVP, i.e., the first non-trivial speedup over the best known algorithm for SVP on general lattices. (In fact, this reduction works for a more general class of lattices—semi-stable lattices with not-too-large λ_1.)

* Due to space constraints, we have omitted some discussion, proofs, and figures from this version of the paper. We strongly encourage the reader to look at the full version, which is available at [7].

** Part of this work was while H.B. was at the University of Michigan and supported by the National Science Foundation under Grant No. CCF-2006857. N.S. was supported in part by the National Science Foundation under Grant No. CCF-2122230. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the National Science Foundation.

2. We show a simple public-key encryption scheme that is secure if (an appropriate variant of) ZSVP 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 ZSVP is hard for *any* input basis, then ZSVP 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 [9]. 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 [15], and they also used this to analyze the security of a public-key encryption scheme. Similar ideas also appeared in [11,20,5] in different contexts.)
4. We perform experiments to determine how practical basis reduction performs on bases of $\mathbb{Z}^n$ that are generated in different ways and how heuristic sieving algorithms perform on $\mathbb{Z}^n$. Our basis reduction experiments complement and add to those performed by Blanks and Miller, as we work with a larger class of algorithms (i.e., larger block sizes) and study the "provably hard" distribution of bases described above. Our sieving experiments confirm that heuristic sieving algorithms perform as expected on $\mathbb{Z}^n$.

1 Introduction

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 [32] 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 $\mathbb{Z}^n$ by simply outputting one of these vectors. 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 shortest vector in this lattice. (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 $\mathbf{B}$.[4] 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 [19,38,17,24,25,12,22]. 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 [2]. So, we do not even know whether $\mathbb{Z}$SVP is *any* easier at all than SVP on arbitrary lattices, let alone whether there exists a polynomial-time algorithm!

1.1 Our results

In this paper, we set aside the (apparently difficult) 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?
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 suitable for use in cryptography?
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.

[4] Of course, this is not actually a hard problem, 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 $\boldsymbol{z} := (59, 396, 225, -326)^T$.

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 GapSVP over general lattices for any constant $\gamma = O(1)$ (where GapSVP is the decision version of SVP in which the goal is simply to determine whether there exists a short vector, rather than to actually find one). In other words, we show that in order to find an exact shortest non-zero vector in a rotation of $\mathbb{Z}^n$, it suffices to simply approximate the length of a shortest non-zero 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 (Informal. See Corollary 2). *There is an efficient reduction from $\mathbb{Z}$SVP to γ-approximate GapSVP (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 $O(1)$-GapSVP, we immediately obtain a $2^{n/2+o(n)}$-time provably correct algorithm for $\mathbb{Z}$SVP [2]. (And, under a purely geometric conjecture, we obtain a running time of $(4/3)^{n+o(n)} \approx 2^{0.415n}$ [37].) See the full version [7] for a discussion of a more general class of lattices to which these results apply.

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

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 feel that it is premature to base the security of real-world cryptography on the hardness of $\mathbb{Z}$SVP and related problems. Indeed, although $\mathbb{Z}$SVP is fairly well-studied, it is not nearly as well-studied as, e.g., (plain) SVP or factoring, and should therefore be treated with more skepticism. Furthermore, there is currently no consensus about whether $\mathbb{Z}$SVP is actually hard among those who study it.

With that said, we show an encryption scheme that is secure if it is difficult to distinguish a rotation of $\mathbb{Z}^n$ *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 2.1 for the definition of the smoothing parameter.) We call these problems γ-$\mathbb{Z}$GapSVP and α-$\mathbb{Z}$GapSPP, respectively.

Theorem 2 (Informal, see Theorem 12). *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* (promise) problems. In particular, our encryption scheme is secure unless there is a polynomial-time algorithm that correctly 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 than γ-GapSVP, or even than γ'-GapSVP. (Indeed, we are 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 $\sqrt{\log n}$ smaller than trivial.

The approximation factor α for $\mathbb{Z}$GapSPP is harder to interpret, but in the full version [7] we include some discussion.

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 [18] 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. (Similar ideas have previously appeared in somewhat different contexts [11,20,5]. In particular, [11] introduced the idea of sampling a "discrete Gaussian basis" from an arbitrary basis. More recently, in independent work that was published on ePrint before this work, [15] used similar ideas in a context very similar to ours. See Section 1.2.)

This gives a theoretically rigorous answer to the question studied by Blanks and Miller [9], who considered the relative hardness of solving $\mathbb{Z}$SVP for different input bases and asked whether there was a clear choice for a how to generate "hardest possible" bases. We show that there is in fact 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.[5] These experiments complement similar experiments performed by Blanks and Miller [9]. 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 [9]. 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 [9] with *much* shorter vectors (which corresponds to a more efficient encryption scheme).

Our second set of experiments document a *threshold* phenomenon that is evident in these basis reduction experiments with $\mathbb{Z}^n$. Specifically, the output of basis reduction algorithms run on bases of $\mathbb{Z}^n$ is almost always an *exact* shortest non-zero vector or a vector much longer than this. I.e., once basis reduction finds a vector in $\mathbb{Z}^n$ whose length is below some threshold, it nearly always simply finds a shortest vector. We document this phenomenon in our context. (After a preliminary version of this paper was released, we learned of a body of work studying this phenomenon in a larger context and providing compelling heuristic explanations of it, such as in [4,13]. See [14, Section 4.2] for more recent experiments, discussion of this phenomenon in the specific context of $\mathbb{Z}^n$, and additional references.)

Our third and final set of experiments studies the performance of a *heuristic sieving algorithm* on $\mathbb{Z}^n$. Specifically, we ran the Gauss sieve, due to Micciancio and Voulgaris [30], on $\mathbb{Z}^n$. In fact, $\mathbb{Z}^n$ is a particularly interesting lattice for heuristic sieving algorithms because $\mathbb{Z}^n$ is known to grossly violate the heuristics that are used to design and analyze these algorithms. (See Section 6.3.) Nevertheless, we confirm that the Gauss sieve performs more-or-less exactly the same on $\mathbb{Z}^n$ as it does on other lattices—in spite of the fact that some of the heuristic justification for the Gauss sieve does not extend to $\mathbb{Z}^n$. To our knowledge, such experiments had not been published before.

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 [19,17,24,25,12].

[5] 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.

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 [38]. 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 $\gamma \leq \sqrt{n}/2$).

Our public-key encryption scheme is quite similar to a scheme recently proposed by Ducas and van Woerden [15], 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 also 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 [15], 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 [15] 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 and van Woerden list many "remarkable" lattices, many of which are better suited to their construction.) 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. Indeed, our scheme is particularly well suited to $\mathbb{Z}^n$ (as we discuss more in the full version [7]). Because of this difference, our scheme achieves security under arguably weaker hardness assumptions. The assumptions are not directly comparable, however, as [15]'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 [11,20,5], though in different contexts.)

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 [9]. Our empirical work on different bases for $\mathbb{Z}^n$ is best viewed as follow-up work to [9]. 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 [9].

Finally, we note that recent follow-up work to this paper [8] has continued the study of the cryptosystem that we propose.

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*. By default logarithms are base e.

We refer the reader to the full version [7] for basic definitions of lattices, the successive minima λ_i, the lattice determinant, the Gram matrix, SVP, GapSVP, and unique SVP.

2.1 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

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

for any (measurable) subset $S \subseteq \mathbb{R}^n$. We simply write D_s for D_s^1.

We prove the following lemma in the full version [7]. It shows that when $\boldsymbol{X}$ is sampled from D_s^n, $\mathrm{dist}(\boldsymbol{X}, \mathbb{Z}^n)$ is highly concentrated.

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

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

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 [10], which is a slight strengthening of a result in [18].

Theorem 3. *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* $D_{\mathcal{L},s}$.[6]

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

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

Lemma 2 ([29, 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 3 ([29, Lemma 3.2]). *For any lattice* $\mathcal{L} \subset \mathbb{R}^n$ *and any* $\varepsilon > 2^{-n}$

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

Lemma 4 ([20, 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 of* $\mathbb{Z}^n$ *except with probability* $2^{-\Omega(n)}$*.*

2.2 Lattice problems

We will use a result of Lyubashevsky and Micciancio that gives an efficient, dimension-preserving reduction from γ-uSVP to γ-GapSVP for polynomially bounded $\gamma = \gamma(n)$.

Theorem 4 ([28, Theorem 3]). *For any* $1 \leq \gamma \leq \mathrm{poly}(n)$*, there is a dimension-preserving Cook reduction from* γ-uSVP *to* γ-GapSVP*.*

We will also make use of the following algorithm.

Theorem 5 ([2, Corollary 6.6]). *There is a* $2^{n/2+o(n)}$*-time algorithm that solves* γ-GapSVP *with* $\gamma = 1.93 + o(1)$*.*

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 with the additional requirement that the input basis $\mathbf{B}$ satisfy $\mathcal{L}(\mathbf{B}) \cong \mathbb{Z}^n$.

Definition 1. *For* $\gamma = \gamma(n) \geq 1$*, 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* $\|\boldsymbol{v}\| \leq \gamma \cdot \lambda_1(\mathcal{L})$*.*

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

[6] In fact, the algorithm even works for any parameter $s \geq \sqrt{\log(2n+4)/\pi} \cdot \max_i \|\widetilde{\boldsymbol{b}}_i\|$, where $\widetilde{\boldsymbol{b}}_i$ is the ith Gram-Schmidt vector of the basis $\mathbf{B}$.

2.3 Primitive vectors and vector counting

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 $\mathcal{L}_{\text{prim}}$ denote the set of primitive vectors in $\mathcal{L}$. For a lattice $\mathcal{L}$ and $r > 0$, let $N(\mathcal{L}, r) := |\{\boldsymbol{x} \in \mathcal{L} : \|\boldsymbol{x}\| \leq r\}|$ and let $N_{\text{prim}}(\mathcal{L}, r) := |\{\boldsymbol{x} \in \mathcal{L}_{\text{prim}} : \|\boldsymbol{x}\| \leq r\}|/2$, where in the latter expression we divide by two so that we effectively count $\pm\boldsymbol{x} \in \mathcal{L}$ as a single vector.

We will use the following bound from [34] on the number of integer points in a ball $r\mathcal{B}_2^n$ for various radii r, where $\mathcal{B}_2^n$ denotes the closed Euclidean unit ball.

Proposition 1 ([34, Claim 8.2]). *For any $n \geq 1$ and any radius $1 \leq r \leq \sqrt{n}$ with $r^2 \in \mathbb{Z}$,*

$$(2n/r^2)^{r^2} \leq |\mathbb{Z}^n \cap r\mathcal{B}_2^n| \leq (2e^3 n/r^2)^{r^2} \ .$$

A lattice $\mathcal{L} \subseteq \mathbb{R}^n$ satisfying $\det(\mathcal{L}') \geq 1$ for all sublattices $\mathcal{L}' \subseteq \mathcal{L}$ is called *semi-stable*. We will also use the following bound from [34] on $|\mathcal{L} \cap r\mathcal{B}_2^n|$ where $\mathcal{L}$ is a semi-stable lattice.

Proposition 2 ([34, Corollary 1.4, Item 1]). *Let $t := 10(\log n + 2)$ and let $\mathcal{L}$ be a semi-stable lattice. Then for any $r \geq 1$, $|\mathcal{L} \cap r\mathcal{B}_2^n| \leq 3e^{\pi t^2 r^2}/2$.*

2.4 Probability

Lemma 5 (Chernoff-Hoeffding bound [21]). *Let $X_1, \ldots, X_M \in [0,1]$ be independent and identically distributed random variables. Then, for $s > 0$,*

$$\Pr\left[\left|M\,\mathbb{E}[X_i] - \sum X_i\right| \geq sM\right] \leq 2e^{-Ms^2/10} \ .$$

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 $\mathbb{Z}^n$ 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." (We do not try to make this very general statement formal. In particular, we do not try to classify the set of computational problems for which this result applies. Instead, we simply provide an example.) Similar ideas appeared in [11,20,5,15].

We say that an algorithm $\mathcal{A}$ that takes as input vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N \in \mathcal{L}$ that form a generating set of a lattice $\mathcal{L}$ and outputs a basis $\mathbf{B}$ of $\mathcal{L}$ is *rotation-invariant* if for any orthogonal transformation $R \in \mathsf{O}_n(\mathbb{R})$, $\mathcal{A}(R\boldsymbol{y}_1, \ldots, R\boldsymbol{y}_N) = R(\mathcal{A}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N))$. For example, the LLL algorithm yields an efficient rotation-invariant algorithm that converts a generating set to a basis, and in Section 3.1 we give a more efficient algorithm that also does this. Given such an $\mathcal{A}$, our distribution is then the following.

Definition 2. *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 $\mathbf{B} := \mathcal{A}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_i)$. If $\mathbf{B} \in \mathbb{Z}^{n \times n}$ 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 4, the above procedure terminates in polynomial time except with negligible probability.

Theorem 6. *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 3 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 4 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).

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

Corollary 1. *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 6 and outputs the result.

Using Corollary 1, we can easily reduce worst-case variants of lattice problems on rotations of $\mathbb{Z}^n$ 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. *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 7. *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 1, 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 oracle on input $\mathbf{B}'$, receiving as output some vector $\boldsymbol{y}' \in \mathcal{L}'$. Finally, it outputs $\boldsymbol{y} := R^{-1}\boldsymbol{y}'$.

3.1 A rotation-invariant generating set to basis conversion algorithm

For completeness, we now specify and analyze a rotation-invariant algorithm (Algorithm 1) for converting a generating set $Y = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$ to a basis. After we published a preliminary version of this work, we learned that Li and Nguyen developed a very similar algorithm in [26, Algorithm B.1], and showed an optimized variant in [27, Section 4].

The algorithm $\mathcal{A}$ itself is perhaps best viewed as a "lazy" variant of the LLL algorithm. In particular, unlike LLL, $\mathcal{A}$ simply works to find *some* basis of the lattice generated by Y, and makes no attempt to further reduce the basis. More quantitatively, in Theorem 8, we upper bound the number of swaps performed by Algorithm 1 for (rotations of) integer lattices by $n \log_2 \beta$, where n is the rank of the input lattice and β is the maximum norm of a vector in the input generating set Y. (It is common in the literature to state the running time of basis reduction algorithms in this form.) For comparison, standard analysis of the LLL algorithm (see, e.g., [33]) upper bounds the number of swaps it performs by $O(n^2 \log \beta)$.

Define the (generalized) Gram-Schmidt vectors corresponding to a sequence $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$ of (not necessarily linearly independent) vectors as follows:

$$\widetilde{\boldsymbol{y}}_1 := \boldsymbol{y}_1 \ ,$$
$$\widetilde{\boldsymbol{y}}_i := \boldsymbol{y}_i - \sum_{\substack{j<i, \\ \widetilde{\boldsymbol{y}}_j \neq \boldsymbol{0}}} \frac{\langle \boldsymbol{y}_i, \widetilde{\boldsymbol{y}}_j \rangle}{\langle \widetilde{\boldsymbol{y}}_j, \widetilde{\boldsymbol{y}}_j \rangle} \widetilde{\boldsymbol{y}}_j \qquad \text{for } i = 2, \ldots, N.$$

We next prove that Algorithm 1 is correct, rotation invariant, and in fact quite efficient. Recall that a generating-set-to-basis conversion algorithm $\mathcal{A}$ being

Algorithm 1: Rotation-Invariant Generating Set to Basis Conversion

Input: A generating set $Y = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N) \in \mathbb{R}^{m \times N}$ of a lattice $\mathcal{L}$ of rank $1 \leq n \leq N$.
Output: A basis of $\mathcal{L}$.

`// Size-reduction step.`
Compute the Gram-Schmidt vectors $\widetilde{\boldsymbol{y}}_1, \ldots, \widetilde{\boldsymbol{y}}_N$ corresponding to $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$.
for $i = 2, \ldots, N$ **do**
  for $j = i-1, \ldots, 1$ *with* $\widetilde{\boldsymbol{y}}_j \neq \boldsymbol{0}$ **do**
    $\boldsymbol{y}_i \leftarrow \boldsymbol{y}_i - \lfloor \mu_{i,j} \rceil \cdot \boldsymbol{y}_j$ // $\mu_{i,j} := \langle \boldsymbol{y}_i, \widetilde{\boldsymbol{y}}_j \rangle / \langle \widetilde{\boldsymbol{y}}_j, \widetilde{\boldsymbol{y}}_j \rangle$.
  end
end
Delete any identically zero columns from Y, and update N to be the new number of columns in Y.

`// Swap step.`
if *there exists* $i \in \{2, \ldots, N\}$ *such that* $\widetilde{\boldsymbol{y}}_i = \boldsymbol{0}$ **then**
  Swap $\boldsymbol{y}_j$ and $\boldsymbol{y}_i$, where $j < i$ is the minimum index such that $\boldsymbol{y}_i \in \mathrm{span}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j)$.
  goto size-reduction step.
end

return Y.

rotation invariant means that for all input generating sets $Y \in \mathbb{R}^{m \times N}$ and $R \in \mathsf{O}_m(\mathbb{R})$, $R\mathcal{A}(Y) = \mathcal{A}(RY)$.

Theorem 8. *On input a generating set $Y = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N) \in \mathbb{R}^{m \times N}$ of a lattice $\mathcal{L}$ of rank $n \geq 1$, Algorithm 1 outputs a basis of $\mathcal{L}$. Furthermore, Algorithm 1 is rotation invariant and performs at most $n \log_2 \beta - \log \det(\mathcal{L})$ swap operations, where $\beta := \max_{i \in \{1,\ldots,N\}} \|\boldsymbol{y}_i\|$. In particular, if $\mathcal{L}$ is (a rotation of an) integer lattice then $\det(\mathcal{L}) \geq 1$ and so Algorithm 1 performs at most $n \log_2 \beta$ swaps.*

Proof. In the full version [7], we include a (straightforward) proof that Algorithm 1 does in fact output a basis and is in fact rotation invariant.

It remains to upper bound the number of swaps performed by Algorithm 1. Define the potential function

$$P(Y) := \prod_{\substack{i \in \{1,\ldots,N\}, \\ \widetilde{\boldsymbol{y}}_i \neq \boldsymbol{0}}} \|\widetilde{\boldsymbol{y}}_i\| \ ,$$

and note that $P(Y)$ is equal to the determinant of the sublattice of $\mathcal{L}$ spanned by vectors $\boldsymbol{y}_i$ with $\widetilde{\boldsymbol{y}}_i \neq \boldsymbol{0}$. Therefore, because the algorithm maintains the invariant that Y is a generating set of $\mathcal{L}$, we have that $P(Y) \geq \det(\mathcal{L})$. Using the same invariant, we also have that at each iteration there are exactly n vectors with non-zero Gram-Schmidt vectors. So, by definition of β, the input generating set

$Y_0 = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$ satisfies

$$P(Y_0) = \prod_{\substack{i \in \{1,\ldots,N\}, \\ \widetilde{\boldsymbol{y}}_i \neq \boldsymbol{0}}} \|\widetilde{\boldsymbol{y}}_i\| \leq \prod_{\substack{i \in \{1,\ldots,N\}, \\ \widetilde{\boldsymbol{y}}_i \neq \boldsymbol{0}}} \|\boldsymbol{y}_i\| \leq \beta^n \ . \tag{1}$$

Finally, we show that $P(Y)$ decreases by a multiplicative factor of at least 2 after each swap operation. Let $Y = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$ and $Y' = (\boldsymbol{y}'_1, \ldots, \boldsymbol{y}'_N)$ denote the respective generating sets in Algorithm 1 before and after performing a given swap operation on $\boldsymbol{y}_j$ and $\boldsymbol{y}_i$ for $j < i$.

We claim that $\widetilde{\boldsymbol{y}}'_k = \widetilde{\boldsymbol{y}}_k$ for all $k \neq j$. This is immediate for $k < j$ because $\boldsymbol{y}'_k = \boldsymbol{y}_k$ for such k. For $k > j$, it follows by noting that $\mathrm{span}(\boldsymbol{y}'_1, \ldots, \boldsymbol{y}'_j) = \mathrm{span}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j)$, which in turn follows by noting that, by the algorithm's choice of i and j, $\boldsymbol{y}'_j = \boldsymbol{y}_i$ and $\boldsymbol{y}_i \in \mathrm{span}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j) \setminus \mathrm{span}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{j-1})$. Furthermore, $\boldsymbol{y}_i \in \mathrm{span}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j) \setminus \mathrm{span}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{j-1})$ implies that $\widetilde{\boldsymbol{y}}_j$ is non-zero.

Let π_k denote projection onto $\mathrm{span}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k)^\perp$. We then have that

$$\frac{P(Y')}{P(Y)} = \prod_{\substack{k \in \{1,\ldots,N\}, \\ \widetilde{\boldsymbol{y}}_k \neq \boldsymbol{0}}} \frac{\|\widetilde{\boldsymbol{y}}'_k\|}{\|\widetilde{\boldsymbol{y}}_k\|} = \frac{\|\widetilde{\boldsymbol{y}}'_j\|}{\|\widetilde{\boldsymbol{y}}_j\|} = \frac{\|\pi_{j-1}(\boldsymbol{y}_i)\|}{\|\widetilde{\boldsymbol{y}}_j\|} = \frac{|\mu_{i,j}| \cdot \|\widetilde{\boldsymbol{y}}_j\|}{\|\widetilde{\boldsymbol{y}}_j\|} \leq 1/2 \ .$$

The final equality again uses the fact that $\boldsymbol{y}_i \in \mathrm{span}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j)$, and the inequality holds because $\mu_{i,j} := \langle \boldsymbol{y}_i, \widetilde{\boldsymbol{y}}_j \rangle / \langle \widetilde{\boldsymbol{y}}_j, \widetilde{\boldsymbol{y}}_j \rangle$ has magnitude at most $1/2$ after the size-reduction step.

Therefore, by Equation (1), Algorithm 1 performs at most

$$\log_2(P(Y_0)/\det(\mathcal{L})) \leq n \log_2 \beta - \log\det(\mathcal{L})$$

swap operations, as needed.

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. (See also [8] for follow-up work implementing the scheme and studying its security.)

The encryption scheme itself is described below. There are public parameters $s > 0$ and $r > 0$, which are functions of the security parameter n (i.e., $s = s(n)$ and $r = r(n)$). In particular, the parameter s will control the length of the basis used as the public key, and the parameter r is a noise parameter. In the full version [7], we provide more discussion of these parameters.

- $\mathsf{Gen}(1^n)$: 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 $\mathbb{Z}^n$. Run Algorithm 1[7] on input $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ to obtain a basis $\mathbf{B}$ of $\mathbb{Z}^n$ and let $\boldsymbol{G} := \mathbf{B}^T\mathbf{B}$. Output $sk := \mathbf{B}$ and $pk := \boldsymbol{G}$.

[7] One can instead run any rotation-invariant algorithm that converts generating sets into bases, as defined in Section 3. We simply suggest Algorithm 1 for concreteness.

- $\mathsf{Enc}(pk, b \in \{0,1\})$:
  - If $b = 0$, sample $\boldsymbol{X} \in \mathbb{R}^n$ from a continuous Gaussian distribution with probability density function

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

    and output $\boldsymbol{c} := \boldsymbol{X} \bmod 1$ (i.e., the coordinates of $\boldsymbol{c}$ are the fractional parts of the coordinates of $\boldsymbol{X}$).
  - If $b = 1$, 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, where

$$d := \frac{n}{12} - \frac{\exp(-\pi r^2)}{2\pi^2} \cdot n \ .$$

We first concern ourselves with the correctness of this scheme. In particular, the following lemma tells us that the decryption algorithm will answer correctly except with probability roughly $\exp(-e^{-\pi r^2} n)$. In order to be conservative, we will want to take r to be as big as possible, so we will take r to be slightly smaller than $\sqrt{\log n/\pi}$. E.g., we can take $r = \sqrt{\log n/(10\pi)}$. This is the maximal choice for r up to a constant, since if we took, e.g, $r \geq \sqrt{\log n}$, then ciphertexts of zero would be statistically close to ciphertexts of one, making decryption failures unreasonably common.

Lemma 6. *For $r \geq 1$, let $\delta := \exp(-\pi r^2)$. 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 5) that

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

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 1,

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

where $\nu := \frac{n}{12} - \frac{\delta}{\pi^2} \cdot n$, and $\varepsilon := \delta^4/3$. Notice that $\frac{d-\nu-\varepsilon n}{n} = \frac{\delta}{2\pi^2} - \delta^4/3 > \delta/100$. 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.

Definition 4. *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 $\|\boldsymbol{b}_i\| \leq \rho$ 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)-ℤGvU is hard *if no probabilistic polynomial-time algorithm $\mathcal{A}$ can solve this problem with probability better than $1/2 + \mathrm{negl}(n)$.*

Theorem 9. *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 ℤGvU 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 6 with Algorithm 1 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, ℤGvU 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 5. *For any approximation factor $\alpha = \alpha(n) \geq 1$, $\varepsilon \in (0, 1/2)$, and a length bound $\rho = \rho(n) > 0$, the problem $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}$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 $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}$GapSPP is hard, then our encryption scheme with $r := \sqrt{\log n/(10\pi)}$ is secure for any $\varepsilon < n^{-\omega(1)}$ and $\alpha \leq \eta_\varepsilon(\mathbb{Z}^n)/r \approx \sqrt{10\log(n/\varepsilon)/\log n} \approx \sqrt{\log(1/\varepsilon)/\log n}$.

Theorem 10. *For any efficiently computable $\varepsilon = \varepsilon(n) \in (0, 1/2)$ and integer $\ell = \ell(n) \geq 100n/(\delta - \varepsilon)^2$, there is a reduction from $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}$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 5),

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

as needed.

On the other hand, in the NO case, by Lemma 2, $\boldsymbol{y}_i$ 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, $\Pr[b_i^* = b_i] \leq 1/2 + \varepsilon$, and again these events are independent. Therefore, by the Chernoff-Hoeffding bound again,

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

as needed.

(Note that the following definition is not simply the restriction of GapSVP to rotations $\mathcal{L}$ of $\mathbb{Z}^n$—which would be a meaningless problem since all such $\mathcal{L}$ have $\lambda_1(\mathcal{L}) = 1$. Instead, it is the problem of distinguishing $\mathbb{Z}^n$ from a lattice $\mathcal{L}$ with

significantly larger $\lambda_1(\mathcal{L}^*)$. Of course, since $\mathbb{Z}^n$ is self dual, and since one can efficiently test whether a lattice is self dual, we could without loss of generality restrict our attention to self-dual lattices and then equivalently work with $\lambda_1(\mathcal{L})$ instead of $\lambda_1(\mathcal{L}^*)$.)

Definition 6. *For parameters $\rho = \rho(n) > 0$ and $\gamma = \gamma(n) \geq 1$, the problem (ρ,γ)-$\mathbb{Z}$GapSVP 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 $\lambda_1(\mathcal{L}^*) > \gamma$.*

Theorem 11. *For any $\varepsilon = \varepsilon(n)$ with $2^{-n} < \varepsilon < 1/2$, $\rho = \rho(n) > 0$, and $\gamma = \gamma(n) \geq 10\sqrt{n/\log(n/\varepsilon)}$, there is an efficient reduction from (ρ,γ)-$\mathbb{Z}$GapSVP to $(\alpha,\varepsilon,\rho)$-$\mathbb{Z}$GapSPP for $\alpha := \gamma\sqrt{\log(n/\varepsilon)/n}/10$.*

Proof. The reduction simply calls its $\mathbb{Z}$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 3 if $\lambda_1(\mathcal{L}^*) > \gamma$, then $\eta_\varepsilon(\mathcal{L}) < \sqrt{n}/\gamma \leq 10\sqrt{n/\log(n/\varepsilon)} \cdot \eta_\varepsilon(\mathbb{Z}^n)/\gamma = \eta_\varepsilon(\mathbb{Z}^n)/\alpha$, so that the oracle must output NO.

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 $\mathbb{Z}$GapSVP (or even $\mathbb{Z}$GapSPP) is hard.

Theorem 12. *Let $r := \sqrt{\log n/(10\pi)}$, and let d be as in Lemma 6. 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 $(\alpha,\varepsilon,\rho)$-$\mathbb{Z}$GapSPP is hard for $\alpha := \eta_\varepsilon(\mathbb{Z}^n)/r \approx \sqrt{10\log(n/\varepsilon)/\log n}$ and $\rho := s/\sqrt{(\log 2n + 4)/\pi}$* or *if (ρ,γ)-$\mathbb{Z}$GapSVP is hard for $\gamma := 10\sqrt{n/\log(n/\varepsilon)} \cdot \alpha \approx \sqrt{10n/\log n}$.*

5 Reductions and provable algorithms

In this section, we give a reduction 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 uSVP, we show that for any constant $\varepsilon > 0$ there is a $2^{n/2+o(n)}$-time algorithm for $\mathbb{Z}$SVP.[8] This improves exponentially over the fastest known algorithm for SVP on general lattices [2], which runs in $2^{n+o(n)}$ time and was previously the fastest known algorithm even for the special case of $\mathbb{Z}$SVP. In fact, our $2^{n/2+o(n)}$-time algorithm works more generally for semi-stable lattices whose minimum distance is not too large.

We note that our reduction is similar to the reduction from SVP to uSVP in [36] though it works in a very different regime (we solve *exact* $\mathbb{Z}$SVP using

[8] We note again in passing that under a purely geometric conjecture we would in fact obtain a running time of $(4/3)^{n+o(n)} \approx 2^{0.415n}$ [37].

a γ-uSVP oracle for any constant γ, while [36] solves *approximate* SVP using a γ-uSVP oracle for $\gamma \leq 1 + O(\log n/n)$).

Interpreted differently, our 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 The main reduction and algorithms

We next present our main reduction, from which we get our main algorithms.

Sampling using a γ-uSVP oracle Our reduction crucially uses the following theorem, which shows how to use a γ-uSVP oracle to sample short primitive vectors. It is very similar to results in [1,35], but those results are in a slightly different form from what we need. See the full version of the paper [7] for a proof.

Theorem 13. *For any $\gamma = \gamma(n) \geq 1$ and $r > 0$, there is a polynomial-time randomized algorithm with access to a γ-uSVP oracle that takes as input (a basis of a) lattice $\mathcal{L}$ and an integer $A' \geq A := N_{prim}(\mathcal{L}, \gamma r)$ and outputs a vector $\boldsymbol{y} \in \mathcal{L}$ such that if $\boldsymbol{x} \in \mathcal{L}$ is a primitive vector with $\|\boldsymbol{x}\| \leq r$ then*

$$\Pr[\boldsymbol{y} = \boldsymbol{x}] \geq \frac{1}{200A' \log(100A')} \ .$$

Furthermore, the algorithm makes a single query to its γ-uSVP oracle on a full-rank sublattice of $\mathcal{L}$.

We emphasize that Theorem 13 holds for any $r > 0$, but that r need not be provided as input.

The main reduction We now present our main reduction. 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 by making roughly A/G queries to a γ-uSVP oracle (and which uses roughly A/G time overall), where $A := N_{\mathrm{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L}))$ and $G := N_{\mathrm{prim}}(\mathcal{L}, \lambda_1(\mathcal{L}))$.[9]

Theorem 14. *Let $\gamma = \gamma(n) \geq 1$ and let $\mathcal{L}$ be a lattice of dimension 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 Turing reduction from (exact)* SVP *on $\mathcal{L}$ to γ-*uSVP *that makes $(A/G) \cdot \mathrm{poly}(n)$*

[9] 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. We note in passing that $2G$ is the so-called kissing number of $\mathcal{L}$.

queries to its γ-uSVP oracle, runs in $(A/G) \cdot \mathrm{poly}(n)$ time overall, and makes all oracle queries on full-rank sublattices of $\mathcal{L}$. In particular, the reduction is dimension-preserving.

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 we can solve SVP on $\mathcal{L}$ using $N_{\mathrm{prim}}(\mathcal{L}, 2^{n/2} \cdot \lambda_1(\mathcal{L})) \cdot \mathrm{poly}(n)$ queries to a $2^{n/2}$-uSVP oracle and in $N_{\mathrm{prim}}(\mathcal{L}, 2^{n/2} \cdot \lambda_1(\mathcal{L})) \cdot \mathrm{poly}(n)$ time overall. But, because the $2^{n/2}$-uSVP oracle can be instantiated with a $\mathrm{poly}(n)$-time algorithm (the LLL algorithm [23]), this implies that there is an algorithm that solves SVP on $\mathcal{L}$ and runs in $N_{\mathrm{prim}}(\mathcal{L}, 2^{n/2} \cdot \lambda_1(\mathcal{L})) \cdot \mathrm{poly}(n)$ time (without using any oracles), and therefore an algorithm that runs in $N_{\mathrm{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L})) \cdot \mathrm{poly}(n)$ time and has access to a γ-uSVP oracle for any $\gamma > 2^{n/2}$.

The reduction from SVP on $\mathcal{L}$ to γ-uSVP for $\gamma \leq 2^{n/2}$ works as follows:

1. Guess G' satisfying $G/2 \leq G' \leq G$, and guess A' satisfying $A \leq A' \leq 2A$.
2. Sample $K := \lceil 200A' \log(100A')/G' \rceil \cdot n$ vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K$ using the algorithm in Theorem 13 with (a basis of) $\mathcal{L}$ and A' as input.
3. Return a shortest vector among the vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K$.

Due to space constraints, we defer proving correctness and performing runtime analysis to the full versionof the paper [7].

Algorithms from Theorem 14 Let $T_{\mathrm{uSVP}}(\gamma, n)$ denote the fastest runtime of a (possibly randomized) algorithm for γ-uSVP on lattices of dimension n. By combining the reduction in Theorem 14, the point counting bound for $\mathbb{Z}^n$ in Proposition 1, the reduction from approximate uSVP to approximate GapSVP from Theorem 4, and the algorithm for $(1.93 + o(1))$-uSVP from Theorem 5 we get the following algorithmic result for $\mathbb{Z}$SVP.

Corollary 2. *For $1 \leq \gamma \leq \sqrt{n}$, there is a randomized algorithm that solves $\mathbb{Z}$SVP on lattices of dimension n in $(2e^3 n/\gamma^2)^{\gamma^2} \cdot T_{\mathrm{uSVP}}(\gamma, n) \cdot \mathrm{poly}(n)$ time. In particular, there is a randomized algorithm that solves $\mathbb{Z}$SVP on lattices $\mathcal{L}$ of dimension n in $2^{n/2+o(n)}$ time.*

Proof. By the rotational invariance of the ℓ_2 norm and Proposition 1,

$$A := N_{\mathrm{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L})) = N_{\mathrm{prim}}(\mathbb{Z}^n, \gamma \cdot \lambda_1(\mathbb{Z}^n)) \leq N(\mathbb{Z}^n, \gamma) \leq (2e^3 n/\gamma^2)^{\gamma^2} \ .$$

The main result then follows immediately from Theorem 14.

The $2^{n/2+o(n)}$-time algorithm for $\mathbb{Z}$SVP follows by instantiating the main result with $T_{\mathrm{uSVP}}(1.93+o(1), n) \leq 2^{n/2+o(n)}$, which follows by combining the fast algorithm for $(1.93+o(1))$-GapSVP from Theorem 5 with the efficient dimension-preserving reduction from uSVP to GapSVP in Theorem 4.

We again emphasize that the $2^{n/2+o(n)}$-time algorithm in Corollary 2 substantially improves over the $2^{n+o(n)}$-time SVP algorithm for general lattices from [2], which was also the previous fastest known algorithm for $\mathbb{Z}$SVP.

In fact, Theorem 14 leads to a $2^{n/2+o(n)}$-time algorithm for SVP on a much larger class lattices than rotations of $\mathbb{Z}^n$, namely, on *semi-stable* lattices $\mathcal{L}$ with $\lambda_1(\mathcal{L})$ not too large. (Recall that a semi-stable lattice $\mathcal{L}$ is one with $\det(\mathcal{L}') \geq 1$ for all sublattices $\mathcal{L}' \subseteq \mathcal{L}$.) Namely, combining Theorem 14 with the point-counting bound for semi-stable lattices in Proposition 2 gives such an algorithm.

Corollary 3. *Let $\gamma = \gamma(n) \geq 1$ and let $t := 10(\log n + 2)$. There is a randomized algorithm that solves* SVP *on semi-stable lattices $\mathcal{L}$ of dimension n in $(3e^{\pi t^2(\gamma \cdot \lambda_1(\mathcal{L}))^2}/2) \cdot T_{\mathrm{uSVP}}(\gamma, n) \cdot \mathrm{poly}(n)$ time. In particular, there is a randomized algorithm that solves* SVP *on semi-stable lattices of dimension n with $\lambda_1(\mathcal{L}) \leq o(\sqrt{n}/\log n)$ in $2^{n/2+o(n)}$ time.*

Proof. The main result follows by plugging $r := \gamma \cdot \lambda_1(\mathcal{L})$ into Proposition 2 to upper bound $A := N_{\mathrm{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L}))$ and then invoking Theorem 14. The $2^{n/2+o(n)}$-time algorithm for semi-stable lattices of dimension n with $\lambda_1(\mathcal{L}) \leq o(\sqrt{n}/\log n)$ follows by noting that, if $\gamma = O(1)$ (in particular, if $\gamma = 1.93+o(1)$), then $e^{\pi t^2(\gamma \cdot \lambda_1(\mathcal{L}))^2}/2 = 2^{o(n)}$. Indeed, the claim then follows by again using the fact that $T_{\mathrm{uSVP}}(1.93 + o(1), n) \leq 2^{n/2+o(n)}$.

We note that Theorem 14 and Corollaries 2 and 3 answer a special case of an interesting question of Ducas and van Woerden [15], 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. Semi-stable lattices $\mathcal{L}$ are $\Omega(\sqrt{n}/\lambda_1(\mathcal{L}))$-unusual in this sense (in particular, rotations of $\mathbb{Z}^n$ are $\Theta(\sqrt{n})$-unusual), and so we answer a special case of this question. Our results *do not* hold for f-unusual lattices more generally, essentially because a lattice that is loose with Minkowski's Theorem may nevertheless have a dense sublattice (i.e., may not be semi-stable).

Hardness from Theorem 14 Corollaries 2 and 3 combine the reduction in Theorem 14 with algorithms for γ-uSVP to get algorithms for SVP on rotations of $\mathbb{Z}^n$ and certain semi-sstable lattices. However, interpreting the reduction in the other direction—assuming that SVP on rotations of $\mathbb{Z}^n$ and certain semi-stable lattices is hard—leads to new *hardness results* 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 $\gamma \geq 1$. This is notable because 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 [3,36]. Similarly, if one assumes that there is no randomized quasipolynomial-time algorithm for SVP on stable lattices with sufficiently small minimum distance then there is also no randomized quasipolynomial-time algorithm for solving γ-uSVP for any quasipolynomial γ.

We also 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 $\mathrm{dist}(\boldsymbol{t}, \mathcal{L}) \leq \alpha \cdot \lambda_1(\mathcal{L})$ as input, the goal is to output a closest lattice point to $\boldsymbol{t}$ (i.e., $\boldsymbol{x} \in \mathcal{L}$ satisfying $\|\boldsymbol{t} - \boldsymbol{x}\| = \mathrm{dist}(\boldsymbol{t}, \mathcal{L})$).

Corollary 4. *The following hardness results hold for* γ-uSVP *and* α-BDD*:*

1. *If there is no randomized* $\mathrm{poly}(n)$*-time algorithm for* $\mathbb{Z}$SVP*, then there is no randomized* $\mathrm{poly}(n)$*-time algorithm for* γ-uSVP *for any constant* $\gamma \geq 1$ *or for* α-BDD *for any constant* $\alpha > 0$.
2. *If there is no randomized* $2^{\mathrm{poly}(\log n)}$*-time algorithm for* SVP *on stable lattices* $\mathcal{L}$ *with* $\lambda_1(\mathcal{L}) \leq \mathrm{poly}(\log n)$*, then there is no randomized* $2^{\mathrm{poly}(\log n)}$*-time algorithm for* γ-uSVP *for any* $\gamma \leq 2^{\mathrm{poly}(\log n)}$ *or for* α-BDD *for any* α *with* $(1/\alpha) \leq 2^{\mathrm{poly}(\log n)}$.

Proof. The contrapositive of the claims for uSVP follow immediately from Corollaries 2 and 3. The claims for BDD follow from this by additionally noting that [28] gives an efficient reduction from γ-uSVP to $(1/\gamma)$-BDD for any $\gamma = \gamma(n) \leq \mathrm{poly}(n)$.

6 Experiments

The code and raw data for our experiments can be found at [6].

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.[10] These results extend those in [9], 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 [16]) 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 often treat LLL as "BKZ with block size 2" (though this is not strictly true). 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, and we report below on the smallest block size that

[10] 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. This does not affect our results because all of our algorithms are invariant under rotation.

		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. 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.

found a shortest non-zero vector in the lattice (where, again, we think of LLL as BKZ with block size 2), if one was found. More data can be found in the associated repository [6].

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.

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. Empirically, we found that this yielded a generating set with high probability. Notice that this is much better than what is proven in Lemma 4. See also [31].

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 10).

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 unreasonably 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. 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.)

Unimodular matrix product sampling. The second basis sampling technique that we analyze was proposed in [9], where it is called Algorithm 3. 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 $\phi(H)$:

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

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, which we call "unimodular matrix product" sampling.

The algorithm takes as input a dimension n, a block size $2 \le d \le n$, an entry magnitude size bound $B \ge 1$, and a word length $L \ge 1$. It then samples L uniformly random matrices $\mathbf{M}_1, \ldots, \mathbf{M}_L$ from $\mathrm{GL}_d(\mathbb{Z}) \cap [-B, B]^{d\times d}$. I.e., each $\mathbf{M}_i$ is sampled from the set of all integer matrices with entries of magnitude at most B and determinant ± 1. Additionally, it samples L uniformly random subsets $K_1, \ldots, K_L \subseteq \{1, \ldots, n\}$ of d indices with $K_i = \{k_1^{(i)}, \ldots, k_d^{(i)}\}$. Finally, it outputs the basis $\mathbf{A} := \prod_{i=1}^{L} \phi_{k_1^{(i)},\ldots,k_d^{(i)}}(\mathbf{M}_i)$. (We also refer the reader to the description of this algorithm in [9, Algorithm 3].)

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 not 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 [9].)

We refer the reader to the full version [7] for discussion of our results and a comparison with those in [9].

Bézout-coefficient-based sampling. We next describe our third basis-sampling algorithm, which was suggested by Joseph Silverman and studied as Algorithm 4 in [9]. The algorithm is based on the following observation. Given the matrix $\mathbf{M} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n-1}) \in \mathbb{Z}^{n\times(n-1)}$, 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

					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

					block size						
n	B	L		d	2	3	4	5	10	20	unbroken
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

					block size						
n	B	L		d	2	3	4	5	10	20	unbroken
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		20
512	1	15360		3	0	0	0	0	0		20
512	1	5120		4	0	0	0	0	0		20

Table 2: Experimental results for basis reduction performed on bases generated using the product of sparse unimodular matrices method described in Section 6.1. The entries under each block size represent the number of times (out of a total of twenty 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.

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.

Indeed, with these observations, this Bézout-coefficient-based sampling algorithm is straightforward to describe. It takes as input a dimension n and an entry magnitude size bound $B \geq 1$. It repeatedly samples a uniformly random matrix $\mathbf{M} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n-1}) \in \{-B, -(B-1), \ldots, B-1, B\}^{n \times (n-1)}$ until the minors of $\mathbf{M}$ of size $n-1$ have no non-trivial common factors. It then uses the extended Euclidean algorithm to compute $\boldsymbol{a}$ such that $\boldsymbol{M}' := (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n-1}, \boldsymbol{a})$ is unimodular, and outputs $\boldsymbol{M}'$. (We also refer the reader to the description of this algorithm in [9, Algorithm 4].) In our experiments, we took $B \in \{1, 10, 100\}$. See Table 3.

We refer the reader to the full version [7] for discussion of minor differences between our implmentation and the implementation in [9].

		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. 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.

Our experiments showed 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 [9].

6.2 A threshold phenomenon

In our data, we noticed a phenomenon. We found that the shortest vector in the bases returned by our basis reduction algorithms almost always had either length one or had length larger than some threshold τ. After a preliminary version of this work was published, we learned about a body of work studying such phenomena and providing compelling heuristic explanations for it. And, Ducas, Postlethwaite, Pulles, and van Woerden did additional experiments shedding much more light on this phenomenon [14].

In an earlier version of this work, we speculated more about the causes of this phenomenon and guessed that the threshold was roughly $\tau \approx \sqrt{n}/2$, but [14] give strong evidence that it actually happens at $\tau \approx \Theta(n)$. We now simply include the results of our experiments in Figure 1 and refer the reader to [14] for more information and additional references.

6.3 Sieving Experiments

Finally, we ran experiments with heuristic sieving on $\mathbb{Z}^n$. In some sense, $\mathbb{Z}^n$ is a particularly interesting lattice for heuristic sieving algorithms because $\mathbb{Z}^n$ vio-

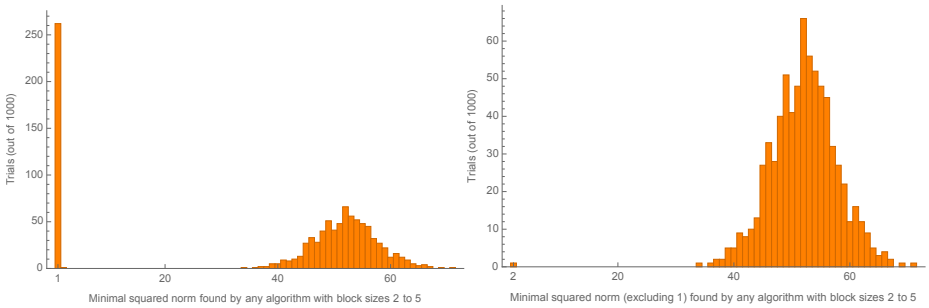

Fig. 1: On the left is a histogram of the squared norm of the shortest vector found by BKZ with block size ≤ 5 for discrete Guassian bases with $n = 128$ and $s = 1000$. On the right is the same histogram without the trials where this norm was 1.

lates the *Gaussian heuristic*, which says that the number of non-zero lattice vectors of length at most r (in a determinant-one lattice) should be approximately equal to the volume of a ball with radius r, which is roughly $(2\pi e r^2/n)^{n/2}$ in large dimensions. Of course, $\mathbb{Z}^n$ completely violates this for small radii. E.g., $\mathbb{Z}^n$ has $2n$ non-zero lattice vectors with length at most 1, while the ball of radius 1 has volume roughly $(2\pi e/n)^{n/2}$, which is much less than one. More generally, for small radii $r \ll \sqrt{n}$, $\mathbb{Z}^n$ has roughly $(Cn/r^2)^{r^2}$ points in a ball of radius r (as in Proposition 1), which is of course much larger than the volume of such a ball.

One might not expect this to cause actual problems for sieving algorithms, but it is worth testing. So, we ran experiments using the Gauss sieve, due to Micciancio and Voulgaris [30], running trials in dimensions $20 \leq n \leq 50$ with Gaussian parameters $s \in \{10, 100, 1000\}$. We ran twenty trials with each pair of values (n, s) (for a total of $20 \cdot 31 \cdot 3 = 1680$ trials). We found that the behavior of this sieving procedure on $\mathbb{Z}^n$ was quite similar to its predicted behavior on lattices that do satisfy the Gaussian heuristic.

Of course, the most important metric of a sieving algorithm is whether it actually finds a shortest non-zero vector. We adopted the common heuristic of running the algorithm until it finds the zero vector (i.e., until there is a collision), and we studied how often the algorithm found a shortest *non-zero* vector before this happened. It would be natural to guess that this should happen in all but a $1/(2n+1)$ fraction of the trials—i.e., we assume that the first vector found with length either 0 or 1 is chosen uniformly at random from the $2n+1$ such vectors. This heuristic matches the data reasonably well.

Next, the number of vectors N sampled by the algorithm (a measure of its space complexity) was well approximated by $N \approx 6.4 \cdot 1.15^n$, as shown in Figure 2. This is completely in line with the predicted behavior of roughly $N = O^*((4/3)^{n/2}) \approx 1.15^n$ (even though this prediction is partially based on a

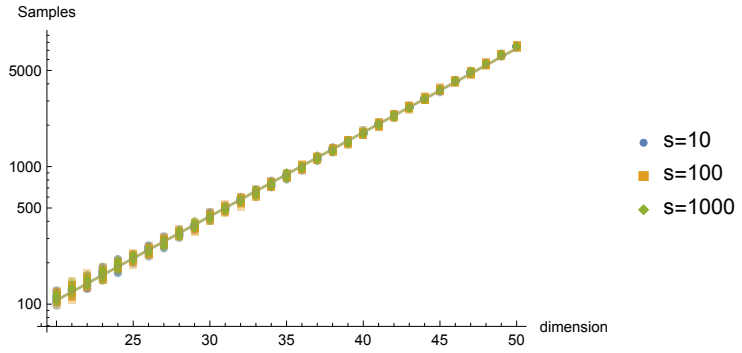

Fig. 2: Scatter plot of the number of vectors sampled by the sieving algorithm in different dimensions with different parameters s, together with the fitted line $6.4 \cdot 1.15^n$. (The fact that the three different parameter values are not distinguishable in the plot reflects the fact that the number of sampled vectors was essentially independent of the parameter size, which is to be expected.)

heuristic that does not directly apply to $\mathbb{Z}^n$), and in line with the numbers reported by Micciancio and Voulgaris and others for sieving experiments on other lattices. So, if sieving algorithms perform differently on $\mathbb{Z}^n$, the difference is rather small. This result did not noticeably depend on the parameter s—i.e. on the lengths of the vectors sampled—which is also what one would expect from a basic heuristic model.

The running time of the algorithm is also well within what we would expect. For example, for parameter $s = 10$, our running times were well approximated by $1.40^n/43000$ seconds (we did not attempt to optimize our code for speed), compared to the expected running time of $O^*((4/3)^n) \approx 1.33^n$, and the running time appears to be proportional to the logarithm of the parameter s, which is again what would be expected. Of course, this running time is subject to many minor implementation details. A less fickle measure is the number of *comparisons* made by the algorithm (i.e., the number of times that the algorithm tests whether subtracting one vector from another will make the latter vector shorter). For this data the simple exponential fit is quite tight and relatively close to what we expect. E.g., for $s = 10$, the number of comparisons is well approximated by $500 \cdot 1.37^n$; for $s = 100$, the fit was $1000 \cdot 1.37^n$; and for $s = 1000$, the fit was $1500 \cdot 1.37^n$. See Figure 3. The slightly larger base of the exponent can likely be explained by lower-order effects, which would require data from a wider range of dimensions to fully explore.

All of the raw data is available at [6].

References

1. Aggarwal, D., Chen, Y., Kumar, R., Li, Z., Stephens-Davidowitz, N.: Dimension-preserving reductions between SVP and CVP in different p-norms. In: SODA

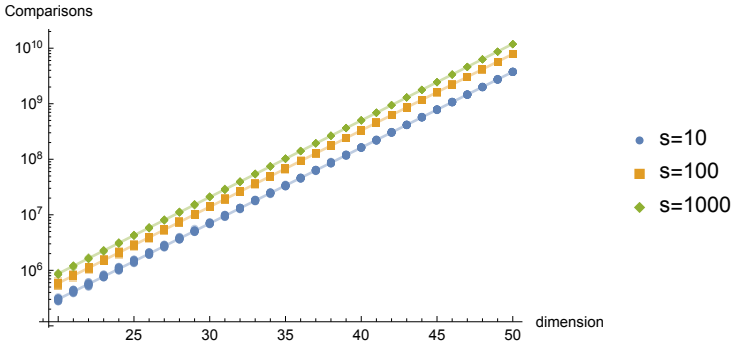

Fig. 3: The number of comparisons made by Micciancio and Voulgaris's Gauss sieve algorithm on $\mathbb{Z}^n$ with different Gaussian parameters s. The trend lines are (roughly) $500 \cdot 1.37^n$, $1000 \cdot 1.37^n$, and $1500 \cdot 1.37^n$ respectively.

(2021) 19
2. Aggarwal, D., Dadush, D., Regev, O., Stephens-Davidowitz, N.: Solving the shortest vector problem in 2^n time using discrete Gaussian sampling. In: STOC (2015) 3, 4, 9, 18, 20
3. Aggarwal, D., Dubey, C.K.: Improved hardness results for unique shortest vector problem. Inf. Process. Lett. **116**(10), 631–637 (2016) 21
4. Albrecht, M.R., Göpfert, F., Virdia, F., Wunderer, T.: Revisiting the expected cost of solving uSVP and applications to LWE. In: ASIACRYPT (2017) 6
5. Aono, Y., Espitau, T., Nguyen, P.Q.: Random lattices: theory and practice, https://espitau.github.io/bin/random_lattice.pdf 2, 5, 7, 10
6. Bennett, H., Ganju, A., Peetathawatchai, P., Stephens-Davidowitz, N.: Experiments on solving SVP on rotations of $\mathbb{Z}^n$. https://github.com/poonpura/Experiments-on-Solving-SVP-on-Rotations-of-Z-n (2021) 22, 23, 28
7. Bennett, H., Ganju, A., Peetathawatchai, P., Stephens-Davidowitz, N.: Just how hard are rotations of $\mathbb{Z}^n$? Algorithms and cryptography with the simplest lattice (2021), https://eprint.iacr.org/2021/1548 1, 4, 5, 7, 8, 13, 14, 19, 20, 24, 25
8. Bennett, H., Little, R.: Revisiting the BGPS rotations-of-$\mathbb{Z}^n$ cryptosystem: An implementation, challenges, and attacks. Preprint. (2023) 8, 14
9. Blanks, T.L., Miller, S.D.: Generating cryptographically-strong random lattice bases and recognizing rotations of $\mathbb{Z}^n$. In: PQCrypto (2021) 2, 5, 6, 7, 8, 22, 24, 25, 26
10. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of Learning with Errors. In: STOC (2013) 9
11. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. J. Cryptol. **25**(4), 601–639 (2012), preliminary version in EUROCRYPT 2010 2, 5, 7, 10
12. Chandrasekaran, K., Gandikota, V., Grigorescu, E.: Deciding orthogonality in Construction-A lattices. SIAM Journal on Discrete Mathematics **31**(2), 1244–1262 (Jan 2017) 3, 6
13. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: LWE with side information: Attacks and concrete security estimation. In: CRYPTO. pp. 329–358. Springer-Verlag (2020) 6

14. Ducas, L., Postlethwaite, E.W., Pulles, L.N., van Woerden, W.: Hawk: Module LIP makes lattice signatures fast, compact and simple. In: Asiacrypt (2023) 6, 26
15. Ducas, L., van Woerden, W.: On the lattice isomorphism problem, quadratic forms, remarkable lattices, and cryptography. In: EUROCRYPT (2022) 2, 5, 7, 10, 21
16. FPLLL development team: fplll, a lattice reduction library, Version: 5.4.1, https://github.com/fplll/fplll, available at https://github.com/fplll/fplll 22
17. Geißler, K., Smart, N.P.: Computing the $M = U^T U$ integer matrix decomposition. In: Cryptography and Coding (2003) 3, 6
18. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC (2008) 5, 9
19. Gentry, C., Szydlo, M.: Cryptanalysis of the revised NTRU signature scheme. In: EUROCRYPT (2002) 3, 6
20. Haviv, I., Regev, O.: On the Lattice Isomorphism Problem. In: SODA (2014) 2, 5, 7, 9, 10
21. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58**, 13–30 (1963) 10
22. Hunkenschröder, C.: Deciding whether a lattice has an orthonormal basis is in co-NP (2019) 3
23. Lenstra, A.K., Lenstra, Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**(4), 515–534 (December 1982) 20
24. Lenstra, H.W., Silverberg, A.: Revisiting the Gentry-Szydlo algorithm. In: CRYPTO (2014) 3, 6
25. Lenstra, H.W., Silverberg, A.: Lattices with symmetry. Journal of Cryptology **30**(3), 760–804 (2017) 3, 6
26. Li, J., Nguyen, P.Q.: Approximating the densest sublattice from Rankin's inequality. LMS J. of Computation and Mathematics **17**(A), 92–111 (2014) 12
27. Li, J., Nguyen, P.Q.: Computing a lattice basis revisited. In: ISAAC (2019) 12
28. Lyubashevsky, V., Micciancio, D.: On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem. In: CRYPTO (2009) 9, 22
29. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. SIAM Journal of Computing **37**(1), 267–302 (2007) 9
30. Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the Shortest Vector Problem. In: SODA (2010) 6, 27
31. Nguyen, P.Q., Pujet, L.: The probability of primitive sets and generators in lattices (2022) 23
32. Peikert, C.: A decade of lattice cryptography. Foundations and Trends in Theoretical Computer Science **10**(4), 283–424 (2016) 2
33. Regev, O.: LLL algorithm. https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/lll.pdf (2004) 12
34. Regev, O., Stephens-Davidowitz, N.: A reverse Minkowski theorem. In: STOC (2017) 10
35. Stephens-Davidowitz, N.: Discrete Gaussian sampling reduces to CVP and SVP. In: SODA (2016) 19
36. Stephens-Davidowitz, N.: Search-to-decision reductions for lattice problems with approximation factors (slightly) greater than one. In: APPROX (2016) 18, 19, 21
37. Stephens-Davidowitz, N.: Lattice algorithms. https://www.youtube.com/watch?v=o4Pl-0Q5-q0 (2020), talk as part of the Simons Institute's semester on lattices. 4, 18
38. Szydlo, M.: Hypercubic lattice reduction and analysis of GGH and NTRU signatures. In: EUROCRYPT (2003) 3, 7