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

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

October 2, 2022

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, what consequences would follow if $\mathbb{Z}$SVP actually is hard? Our results are as follows.

1. We show that $\mathbb{Z}$SVP is in a certain sense strictly easier than SVP on arbitrary lattices. In particular, we show how to reduce $\mathbb{Z}$SVP to an *approximate* version of SVP in the same dimension (in fact, even to approximate *unique* SVP, for any constant approximation factor). Such a reduction seems very unlikely to work for SVP itself, so we view this as a qualitative separation of $\mathbb{Z}$SVP from SVP. As a consequence of this reduction, we obtain a $2^{n/2+o(n)}$-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.)
2. We show a simple public-key encryption scheme that is secure if (an appropriate variant of) $\mathbb{Z}$SVP is actually hard. Specifically, our scheme is secure if it is difficult to distinguish (in the worst case) a rotation of $\mathbb{Z}^n$ from *either* a lattice with all non-zero vectors longer than $\sqrt{n/\log n}$ *or* a lattice with smoothing parameter significantly smaller than the smoothing parameter of $\mathbb{Z}^n$. The latter result has an interesting qualitative connection with reverse Minkowski theorems, which in some sense say that "$\mathbb{Z}^n$ has the largest smoothing parameter."
3. We show a distribution of bases $\mathbf{B}$ for rotations of $\mathbb{Z}^n$ such that, if $\mathbb{Z}$SVP is hard for *any* input basis, then $\mathbb{Z}$SVP is hard on input $\mathbf{B}$. This gives a satisfying theoretical resolution to the problem of sampling hard bases for $\mathbb{Z}^n$, which was studied by Blanks and Miller [BM21]. This worst-case to average-case reduction is also crucially used in the analysis of our encryption scheme. (In recent independent work that appeared as a preprint before this work, Ducas and van Woerden showed essentially the same thing for general lattices [DvW22], and they also used this to analyze the security of a public-key encryption scheme. Similar ideas also appeared in [CHKP12, HR14, AEN] in different contexts.) Along the way to this result, we show a new algorithm for converting a generating set to a basis, which might be of independent interest.
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

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

†Cornell University

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

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. We also observe a threshold phenomenon in which "basis reduction algorithms on $\mathbb{Z}^n$ nearly always find a shortest non-zero vector once they have found a vector with length less than $\sqrt{n}/2$," and we explore this further. Our sieving experiments confirm that heuristic sieving algorithms perform as expected on $\mathbb{Z}^n$.

Contents

1 Introduction

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

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

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

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

Perhaps the simplest example of a lattice is the *integer lattice* $\mathbb{Z}^n$, which has the identity matrix as a basis. Of course, the shortest non-zero vectors in $\mathbb{Z}^n$ are simply the standard basis vectors and their negations $\pm\boldsymbol{e}_1, \ldots, \pm\boldsymbol{e}_n$, which have length one. So, it is trivially easy to find a shortest non-zero vector in $\mathbb{Z}^n$ 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. (E.g., any column of R will do.) But, it does not need to be so easy. For example, the lovely matrix

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

is a basis for a rotation of $\mathbb{Z}^4$, but it is not immediately clear how to find a vector of length one in the lattice generated by $\mathbf{B}$.[1] We write ZSVP 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 ZSVP, leading to a beautiful line of work [GS02, Szy03, GS03, LS14, LS17, CGG17, Hun19]. Frustratingly, despite all of this wonderful work, the fastest known algorithm that is proven to solve ZSVP is still simply the fastest known algorithm that is proven to solve SVP on arbitrary lattices, a $2^{n+o(n)}$-time algorithm [ADRS15]. So, we do not even know whether ZSVP is *any* easier at all than SVP on arbitrary lattices, let alone whether there exists, e.g., a polynomial-time algorithm!

1.1 Our results

In this paper, we set aside the (apparently difficult) question of whether a polynomial-time algorithm for ZSVP exists and instead ask what else we can say about ZSVP. Specifically, we study the following questions.

1. Can we at least solve ZSVP 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?)

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

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.

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 vector in a rotation of $\mathbb{Z}^n$, it suffices to find an *approximate* shortest vector in an arbitrary lattice. (In fact, we reduce to the γ-*unique* Shortest Vector Problem, which is SVP in which the shortest vector is guaranteed to be a factor of γ shorter than "the second shortest vector," appropriately defined.)

Theorem 1.1 (Informal. See Corollary 5.4)**.** *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 [ADRS15]. (And, under a purely geometric conjecture, we obtain a running time of $(4/3)^{n+o(n)} \approx 2^{0.415n}$ [Ste20].) In fact, we show a similar result for exact SVP on a much larger class of lattices, specifically, lattices $\mathcal{L}$ with $\det(\mathcal{L}') \geq 1$ for all sublattices $\mathcal{L}' \subseteq \mathcal{L}$ and with $\lambda_1(\mathcal{L})$ not too large (lattices satisfying the former condition are called *semi-stable*). While this generalized algorithm is not directly related to the remainder of our work, it is related to [DvW22] and makes progress on a question of theirs about "f-unusual lattices"; see Corollary 5.5 and the ensuing discussion.

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 $\mathbb{Z}^n$.

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

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

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

Our reduction can also potentially be viewed as a sort of hardness proof for unique SVP (uSVP), which is a key problem in lattice cryptography. Despite its importance, little is known about its hardness under standard

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

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

Specifically, we show an encryption scheme that is secure if it is difficult to distinguish a rotation 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 2.2 for the definition of the smoothing parameter.) We call these problems γ-$\mathbb{Z}$GapSVP and α-$\mathbb{Z}$GapSPP, respectively.

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

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

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

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

The approximation factor α for $\mathbb{Z}$GapSPP is harder to interpret, in part because computing the smoothing parameter is not nearly as well studied as computing the length of the shortest non-zero vector. (But, see [CDLP13].) However, a recent series of works [DR16, RS17, ERS22, RS22] has shown that there is a certain sense in which "$\mathbb{Z}^n$ has the largest smoothing parameter of any lattice." (E.g., up to a pesky factor of 2, $\mathbb{Z}^n$ is known to have the largest smoothing parameter of any determinant-one lattice whose gram matrix is integral [RS22].) So, there is a certain vague sense in which α-$\mathbb{Z}$GapSPP is the problem of "recognizing $\mathbb{Z}^n$ by one of its most distinguishing features," and we therefore think of it as an approximate analogue of the problem of simply recognizing a rotation of $\mathbb{Z}^n$ (i.e., distinguishing a rotation of $\mathbb{Z}^n$ from any other lattice). See Section 4.4.

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

Sampling provably secure bases. Our next main result, presented in Section 3, is a way to sample a "hardest possible" basis $\mathbf{B}$ for a rotation of $\mathbb{Z}^n$. For example, we show an explicit (efficiently sampleable) distribution of bases $\mathbf{B}$ for rotations of $\mathbb{Z}^n$ such that, if it is hard to solve $\mathbb{Z}$SVP in the worst case, then it is hard to solve $\mathbb{Z}$SVP on input $\mathbf{B}$. The basic idea is to use the discrete Gaussian sampling algorithm of [GPV08] to use any basis of a rotation $\mathcal{L}$ of $\mathbb{Z}^n$ to obtain many discrete Gaussian samples from $\mathcal{L}$, sufficiently many that we have a generating set of $\mathcal{L}$. We can then apply any suitable algorithm that converts a generating set into a basis. (Similar ideas have previously appeared in somewhat different contexts [CHKP12, HR14, AEN]. In particular, [CHKP12] 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, [DvW22] used essentially the same idea as [CHKP12] 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 [BM21], 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.

On the way to this result, we show an algorithm that converts a generating set into a basis that is "rotation-invariant" (in the sense that if the algorithm outputs some basis $\mathbf{B}$ on input Y, then on input RY where $R \in \mathsf{O}_n(\mathbb{R})$ is an orthogonal matrix, the algorithm outputs $R\mathbf{B}$—a property that is crucial for our application). The algorithm is a variant of the LLL algorithm that is faster by a factor of $O(n)$, and to our knowledge it is novel. (Of course, unlike the LLL algorithm, it does not find short lattice vectors.)

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

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

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

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

Our second set of experiments therefore studies this phenomenon specifically. Indeed, we show that the behavior is quite striking. See Section 6.2. While we do offer some rough intuition for why this might happen when one performs basis reduction on bases of $\mathbb{Z}^n$, more study is certainly needed to truly understand this phenomenon.

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 [MV10], 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.

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

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 ℤSVP [GS02, GS03, LS14, LS17, CGG17]. A summary of their results is beyond the scope of this work, but we note that their techniques are very different from those in this work with the exception of Szydlo's heuristic algorithm [Szy03]. In particular, Szydlo presented a heuristic algorithm that solves ℤSVP 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 ℤSVP to $c\sqrt{n}$-SVP. In contrast, we give an efficient reduction with a proof of correctness from ℤSVP to γ-uSVP for any constant γ (and, more generally, a roughly $(n/\gamma^2)^{\gamma^2}$-time reduction for $\gamma \leq \sqrt{n}/2$). Additionally, we note that a similar reduction to the one in Theorem 1.1 appears in a different context in [Ste16b]. There, a sparsification-based reduction from approximate SVP (on general lattices) to approximate uSVP is the key component in a search-to-decision reduction for approximate SVP.

Our public-key encryption scheme is quite similar to a scheme recently proposed by Ducas and van Woerden [DvW22], 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 [DvW22], 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 [DvW22] 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 in Section 4.4. Because of this difference, our scheme achieves security under arguably weaker hardness assumptions (because we work at much larger radii), but each of our ciphertexts encodes just a single-bit plaintext, while [DvW22] encode many plaintext bits in each ciphertext (or, more accurately, they construct a KEM). The assumptions are not directly comparable, however, as [DvW22]'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 [CHKP12, HR14, AEN], though in different contexts. Our proofs in Section 3 immediately generalize to other lattices.)

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

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

Throughout this paper, we work with bases $\mathbf{B}$ that are *rotations* of bases of $\mathbb{Z}^n$, or more precisely, orthogonal transformations of bases of $\mathbb{Z}^n$. And, we sometimes even work with uniformly random orthogonal transformations. We do this largely because it is convenient for our presentation and proofs. Of course, to be fully

formal, we must specify exactly the input format that we use for these bases, which in general will not be rational. Indeed, a true sample from the uniform distribution over orthogonal transformations will not even admit a finite description. We adopt the convention, common in the literature on lattices, of ignoring these issues. They can be resolved by appropriately discretizing the space.

However, there are at least two alternative approaches to using rotations, which avoid the issue of discretization entirely and have some major advantages. In particular, neither approach runs into the representation issues described above.

One alternative approach is to work with the *Gram matrix* $\boldsymbol{G} := \mathbf{B}^T\mathbf{B}$ instead of the basis $\mathbf{B}$ itself. Notice that the Gram matrix is rotation independent—i.e., if $R \in \mathsf{O}_n(\mathbb{R})$ is an orthogonal matrix, then $\mathbf{B}^T\mathbf{B} = (R\mathbf{B})^T R\mathbf{B}$, so that working with the Gram matrix effectively removes the need to discuss rotations. And, with some care, one can move freely between Gram matrices and bases. The Gram matrix is also easy to represent in bits, because the Gram matrix is always an integer matrix when $\mathbf{B}$ generates a rotation of $\mathbb{Z}^n$. In fact, in terms of Gram matrices G, $\mathbb{Z}$SVP admits are particularly elegant description: it is equivalent to the problem of finding an integer matrix $\mathbf{B} \in \mathbb{Z}^{n\times n}$ such that $\boldsymbol{G} = \mathbf{B}^T\mathbf{B}$, given only $\boldsymbol{G}$ (with the promise that such a matrix $\mathbf{B}$ exists). (See, e.g., [BM21].)

In fact, we do explicitly work with the Gram matrix when we present our cryptographic scheme, since in that case we are actually proposing an explicit construction. However, most of the literature (and most of the results from prior work that we rely on) is written in terms of bases, not Gram matrices. So, we (mostly) stick to working with bases.

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

Acknowledgements

The authors would like to thank Divesh Aggarwal, Léo Ducas, Ryan Little, Stephen D. Miller, Shahed Sharif, and Michael Walter for helpful comments. In particular, we thank Léo Ducas for bringing our attention to the question of whether there exist lattices $\mathcal{L}$ in the same genus as $\mathbb{Z}^n$ with large $\lambda_1(\mathcal{L})$ or small $\eta_\varepsilon(\mathcal{L})$ (as discussed in Section 4.5), and Divesh Aggarwal for pointing out the relevance of the $2^{n/2+o(n)}$-time algorithm in [ADRS15]. We also thank Phong Q. Nguyen for sharing an early version of [NP22] with us.

2 Preliminaries

We write I_n for the identity matrix. We write $\mathsf{O}_n(\mathbb{R})$ for the set of all orthogonal linear transformations. That is $\mathsf{O}_n(\mathbb{R})$ is the set of matrices $R \in \mathbb{R}^{n\times n}$ with the property that $R^TR = I_n$. 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.

2.1 Basic lattice definitions

We say that a lattice $\mathcal{L} = \mathcal{L}(\mathbf{B}) \subset \mathbb{R}^n$ with basis $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \mathbb{R}^{n\times n}$ has *dimension* n. A sublattice $\mathcal{L}' \subseteq \mathcal{L}$ of the same dimension is called a *full-rank sublattice*.[3] We use $\lambda_1(\mathcal{L})$ to denote the *minimum distance*

[3] In general, one can consider lattices $\mathcal{L} = \mathcal{L}(\mathbf{B})$ for $\mathbf{B} \in \mathbb{R}^{m\times n}$ with $m > n$. In this case, we refer to m as the (ambient) *dimension* of the lattice, and n as the *rank* of the lattice, and define a full-rank sublattice is one with the same rank n as $\mathcal{L}$. However, throughout this work we assume (essentially without loss of generality) that $m = n$.

of $\mathcal{L}$ (equivalently, the length of the shortest non-zero vector in $\mathcal{L}$). I.e., $\lambda_1(\mathcal{L}) := \min_{\boldsymbol{x}\in\mathcal{L}\setminus\{\mathbf{0}\}}\|\boldsymbol{x}\|$. More generally, we define the *ith successive minimum* $\lambda_i(\mathcal{L})$ for $1 \le i \le n$ (where n is the dimension of the lattice) to be the smallest value of $r > 0$ such that $\mathcal{L}$ contains at least i linearly independent vectors of length r:

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

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

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

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

for some explicit constant $C > 0$.

2.2 The continuous and discrete Gaussian distributions and the smoothing parameter

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

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

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

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

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

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

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

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

where

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

and

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

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

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

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

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

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

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

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

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

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

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

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

2.3 Lattice problems

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

Definition 2.7. *For $n \in \mathbb{Z}^+$ and $\gamma = \gamma(n) \geq 1$, the decision version of the γ-approximate Shortest Vector Problem (γ-GapSVP) is defined as follows. Given a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ of a lattice $\mathcal{L}$ and a value $r > 0$ as input, decide whether*

- *(YES instance) $\lambda_1(\mathcal{L}) \leq r$, or*
- *(NO instance) $\lambda_1(\mathcal{L}) > \gamma r$,*

when one of the two cases is promised to hold.

Definition 2.8. *For $n \in \mathbb{Z}^+$ and $\gamma = \gamma(n) \geq 1$, the unique Shortest Vector Problem with gap γ (γ-uSVP) is the search problem defined as follows. Given a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ of a lattice $\mathcal{L}$ satisfying $\gamma \cdot \lambda_1(\mathcal{L}) < \lambda_2(\mathcal{L})$ as input, output a vector $\boldsymbol{v} \in \mathcal{L}$ with $\|\boldsymbol{v}\| = \lambda_1(\mathcal{L})$.*

We 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 2.9 ([LM09, Theorem 3])**.** *For any $n \in \mathbb{Z}^+$ and $1 \leq \gamma \leq \mathrm{poly}(n)$, there is a dimension-preserving Cook reduction from γ-uSVP to γ-GapSVP.*

We will also make use of the following algorithm. (We thank Divesh Aggarwal for pointing out to us that the below theorem is sufficient for our purposes. Earlier versions of this work used a slower algorithm for γ-SVP.)

Theorem 2.10 ([ADRS15, Corollary 6.6])**.** *There is a $2^{n/2+o(n)}$-time algorithm that solves γ-GapSVP with $\gamma = 1.93 + o(1)$.*

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

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

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

Definition 2.11. *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.

One may also consider the problem of recovering a rotation of $\mathbb{Z}^n$, i.e., of recovering an orthonormal basis of a lattice $\mathcal{L} \cong \mathbb{Z}^n$. This problem is equivalent to the search version of the so-called Lattice Isomorphism Problem (LIP) when one input lattice is fixed to be $\mathbb{Z}^n$. Although we will not explicitly study the problem of recovering rotations of $\mathbb{Z}^n$, it provides the intuitive foundation (though not the formal foundation) for the security of our cryptosystem, i.e., given a public key corresponding to a "bad" basis of $\mathcal{L} \cong \mathbb{Z}^n$, our scheme's security rests intuitively on the assumption that it is hard to find an orthonormal basis of $\mathcal{L}$.

We also note in passing that recovering an orthonormal basis of $\mathcal{L} \cong \mathbb{Z}^n$ is polynomial-time equivalent to $\mathbb{Z}$SVP. Indeed, an orthonormal basis of such a lattice $\mathcal{L}$ in particular contains a shortest non-zero vector in $\mathcal{L}$, and, because $\pi_{\mathrm{span}(\boldsymbol{v}_1)^\perp}(\mathcal{L}) \cong \mathbb{Z}^{n-1}$ for a shortest non-zero vector (i.e., unit-length vector) $\boldsymbol{v} \in \mathcal{L}$, one can find an orthogonal basis of $\mathcal{L}$ by calling a $\mathbb{Z}$SVP oracle, projecting orthogonally to its output, and recursing.

2.4 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 $\mathcal{L}_{\mathrm{prim}}$ denote the set of primitive vectors in $\mathcal{L}$. For a lattice $\mathcal{L}$ and $r > 0$, let $N(\mathcal{L}, r) := |\{\boldsymbol{x} \in \mathcal{L} : \|\boldsymbol{x}\| \leq r\}|$ and let $N_{\mathrm{prim}}(\mathcal{L}, r) := |\{\boldsymbol{x} \in \mathcal{L}_{\mathrm{prim}} : \|\boldsymbol{x}\| \leq r\}|/2$, where the latter expression counts primitive $\pm\boldsymbol{x}$ as a single vector.

We will use the following bound from [RS17] 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. Although we will only need the upper bound, we include a lower bound as well to illustrate that the bound is quite tight. Such bounds were originally shown in [MO90].

Proposition 2.12 ([RS17, 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*. (A semi-stable lattice that additionally satisfies $\det(\mathcal{L}) = 1$ is called *stable*.) We will also use the following bound from [RS17] on $|\mathcal{L} \cap r\mathcal{B}_2^n|$ where $\mathcal{L}$ is a semi-stable lattice. (In fact, [RS17] shows such bounds on the number of lattice points in a shifted ball; we state their result only for the centered ball $r\mathcal{B}_2^n$.)

Proposition 2.13 ([RS17, Corollary 1.4, Item 1])**.** *Let $t := 10(\log n + 2)$ and let $\mathcal{L}$ be a semi-stable lattice. Then for any $r \geq 1$, $|\mathcal{L} \cap r\mathcal{B}_2^n| \leq 3e^{\pi t^2 r^2}/2$.*

2.5 Probability

Lemma 2.14 (Chernoff-Hoeffding bound [Hoe63])**.** *Let $X_1, \ldots, X_M \in [0, 1]$ be independent and identically distributed random variables. Then, for $s > 0$,*

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

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

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

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

3 How to sample a provably secure basis

In this section, we show how to sample a basis $\mathbf{B}$ for a rotation of $\mathbb{Z}^n$ that is "provably at least as secure as any other basis." In particular, we show a distribution of bases $\mathbf{B}$ of rotations of $\mathbb{Z}^n$ that can be sampled efficiently given any basis of a rotation of $\mathbb{Z}^n$ 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 [CHKP12, HR14, AEN, DvW22].

In fact, we give a class of distributions, one for each efficient *rotation-invariant* algorithm that converts a generating set to a basis. We say that an algorithm $\mathcal{A}$ that takes as input vectors $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N \in \mathcal{L}$ 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 $R \in \mathsf{O}_n(\mathbb{R})$, $\mathcal{A}(R\boldsymbol{y}_1, \dots, R\boldsymbol{y}_N) = R(\mathcal{A}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_N))$. (Here, for simplicity, we are assuming that $\mathcal{A}$ is a deterministic algorithm. We could generalize this definition to randomized algorithms and simply require that the distribution of $\mathcal{A}(R\boldsymbol{y}_1, \dots, R\boldsymbol{y}_N)$ be statistically close to the distribution of $R(\mathcal{A}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_N))$.) One can equivalently consider algorithms that work with the gram matrix $\boldsymbol{G} \in \mathbb{R}^{N \times N}$ of the generating set, given by $G_{i,j} := \langle \boldsymbol{y}_i, \boldsymbol{y}_j \rangle$, which is invariant under rotations by construction. 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 3.1. *For any efficient rotation-invariant algorithm $\mathcal{A}$ that converts a generating set to a basis and parameter $s = s(n) \geq 1$ the distribution $(\mathcal{A}, s)$-ZDGS is sampled as follows. For $i = 1, 2, 3, \dots$, sample $\boldsymbol{z}_i \sim D_{\mathbb{Z}^n, s}$. Let $\mathbf{B} := \mathcal{A}(\boldsymbol{z}_1, \dots, \boldsymbol{z}_i)$. 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 2.5, the above procedure terminates in polynomial time except with negligible probability.[5]

Theorem 3.2. *For any efficient rotation-invariant algorithm $\mathcal{A}$ that converts a generating set into a basis, there is an efficient randomized algorithm that takes as input a basis $\mathbf{B} = (\boldsymbol{b}_1, \dots, \boldsymbol{b}_n) \in \mathbb{R}^{n \times n}$ 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 $\mathcal{L}'$*

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

that is distributed exactly as $(\mathcal{A}, s)$-ZDGS together with an orthogonal transformation $R \in \mathsf{O}_n(\mathbb{R})$ such that $R\mathcal{L} = \mathcal{L}'$.

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

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

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

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

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

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

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

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

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

3.1 A rotation-invariant generating set to basis conversion algorithm

In this section we 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. It is similar to the LLL algorithm, but more efficient. 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 3.6, 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. In our main use case, the $\boldsymbol{y}_i$ are sampled from

a discrete Gaussian with parameter s, in which case $\beta \leq \sqrt{n}s$ with high probability.) For comparison, standard analysis of the LLL algorithm (see, e.g., [Reg04]) upper bounds the number of swaps it performs by $O(n^2 \log \beta)$, which depends quadratically rather than linearly on n.

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:

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

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 $\boldsymbol{y}_i \in \operatorname{span}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j)$.
  goto size-reduction step.
end

return Y.

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 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 3.6. *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. We first argue that the output of Algorithm 1 must be a basis of $\mathcal{L}$ assuming that it halts. First, we note that the algorithm preserves Y being a generating set of $\mathcal{L}$ as an invariant. Indeed, this follows from the fact that at each iteration Algorithm 1 only performs size-reduction and swap operations. Additionally, we note that the condition in the 'if' statement in the swap step holds exactly when the vectors in Y are not linearly independent. So, if the algorithm terminates the vectors in Y must be linearly independent and therefore be a basis of $\mathcal{L}$. The fact that Algorithm 1 is rotation-invariant follows from the fact that, if $R \in \mathsf{O}_m(\mathbb{R})$ then the Gram-Schmidt vectors of RY are equal to $R\widetilde{\boldsymbol{y}}_1, \ldots, R\widetilde{\boldsymbol{y}}_N$. In particular, the values $\mu_{i,j}$ are the same for Y and RY.

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

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

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

$$P(Y_0) = \prod_{\substack{i \in \{1,\dots,N\}, \\ \widetilde{\boldsymbol{y}}_i \neq \boldsymbol{0}}} \|\widetilde{\boldsymbol{y}}_i\| \leq \prod_{\substack{i \in \{1,\dots,N\}, \\ \widetilde{\boldsymbol{y}}_i \neq \boldsymbol{0}}} \|\boldsymbol{y}_i\| \leq \beta^n \ . \tag{2}$$

Finally, we show that $P(Y)$ decreases by a multiplicative factor of at least 2 after each swap operation. Let $Y = (\boldsymbol{y}_1, \dots, \boldsymbol{y}_N)$ and $Y' = (\boldsymbol{y}'_1, \dots, \boldsymbol{y}'_N)$ 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 $\operatorname{span}(\boldsymbol{y}'_1, \dots, \boldsymbol{y}'_j) = \operatorname{span}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_j)$, which in turn follows by noting that, by the algorithm's choice of i and j, $\boldsymbol{y}'_j = \boldsymbol{y}_i$ and $\boldsymbol{y}_i \in \operatorname{span}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_j) \setminus \operatorname{span}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_{j-1})$. Furthermore, $\boldsymbol{y}_i \in \operatorname{span}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_j) \setminus \operatorname{span}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_{j-1})$ implies that $\widetilde{\boldsymbol{y}}_j$ is non-zero.

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

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

The final equality again uses the fact that $\boldsymbol{y}_i \in \operatorname{span}(\boldsymbol{y}_1, \dots, \boldsymbol{y}_j)$, 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 (2), 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.

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. As we will discuss in more detail below, choosing s to be smaller decreases the public key size but potentially harms security (though, to our knowledge, the scheme retains its security as long as $s \gtrsim 10$, as we discuss Section 6.1.1; on the other hand, there is no need to take $s > 2^n$, because an adversary can always use the LLL algorithm to reduce to the case when $s \leq 2^n$). Choosing r to be smaller decreases the probability of decryption failures but again potentially harms security (though, again, we do not know of attacks that exploit small choices of r, unless r is chosen to be very small, e.g., $r \ll 1$).

- $\mathsf{Gen}(1^n)$: Sample vectors $\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \dots$ independently from $D_{\mathbb{Z}^n, s}$ until $\boldsymbol{z}_1, \dots, \boldsymbol{z}_k$ generate $\mathbb{Z}^n$. Run Algorithm 1[6] on input $\boldsymbol{z}_1, \dots, \boldsymbol{z}_k$ to obtain a basis $\mathbf{B}$ of $\mathbb{Z}^n$ and let $\boldsymbol{G} := \mathbf{B}^T\mathbf{B}$. Output $sk := \mathbf{B}$ and $pk := \boldsymbol{G}$.

[6] One can also 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[7]
$$\frac{\det(\boldsymbol{G})^{1/2}}{r^n} \cdot \exp(-\pi \boldsymbol{X}^T \boldsymbol{G} \boldsymbol{X}/r^2) = \frac{\det(\mathbf{B})}{r^n} \cdot \exp(-\pi \boldsymbol{X}^T \boldsymbol{G} \boldsymbol{X}/r^2) \ ,$$
    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 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 4.1. *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 2.14) that

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

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 2.1,

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

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. □

[7] This is the probability density function of the continuous n-dimensional Gaussian distribution with mean $\mathbf{0}$ and covariance $r^2/(2\pi) \cdot \boldsymbol{G}^{-1}$. Notice that we would get an identical distribution if we were to sample $\boldsymbol{Y} \sim D_r^n$ and set $\boldsymbol{X} := \mathbf{B}^{-1}\boldsymbol{Y}$.

4.1 Basic security

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

Here and below, we have an additional parameter ρ, which is a bound on the lengths of the input basis vectors. If we set $s = 2^n$ in our encryption scheme, then we could remove ρ by using the LLL algorithm, as we did in Section 3. However, we choose to keep the parameter ρ to allow for the possibility of smaller choices of s, which yield smaller entries in the Gram matrix $\boldsymbol{G}$ and therefore a smaller public key. As far as we know, the hardness of the problem is essentially independent of ρ as long as, e.g., $\rho \gtrsim 100\sqrt{n}$.

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

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

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

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

The algorithm $\mathcal{E}$ takes as input a basis $\mathbf{B} \in \mathbb{R}^{n \times n}$ generating a lattice $\mathcal{L}$, and $\boldsymbol{y} \in [0,1)^n$. It then uses the procedure from Theorem 3.2 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, $\mathbb{Z}$GvU is a rather artificial problem. Below, we show reductions to it from worst-case problems that ask us to distinguish $\mathbb{Z}^n$ from a lattice that is different from $\mathbb{Z}^n$ in a specific way. These can be thought of as "$\mathbb{Z}^n$ versions" of the traditional worst-case lattice problems GapSPP and GapSVP.

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

Definition 4.4. *For any approximation factor $\alpha = \alpha(n) \ge 1$, $\varepsilon \in (0, 1/2)$, 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 $\|\boldsymbol{b}_i\| \le \rho$. 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 $\alpha \le \eta_\varepsilon(\mathbb{Z}^n)/r \approx \sqrt{10\log(n/\varepsilon)/\log n}$.

Theorem 4.5. *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 2.14),

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

as needed.

On the other hand, in the NO case, by Lemma 2.3, $\boldsymbol{y}_i$ is within statistical distance ε of a uniformly random element in $[0, 1)^n$. It follows that, regardless of the behavior of the oracle, for each i, $\Pr[b_i^* = b_i] \leq 1/2 + \varepsilon$, 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. □

Notice that the following definition works with the dual lattice $\mathcal{L}^*$ for convenience. Since $\mathbb{Z}^n$ is self-dual, the problem is equivalent if we replace the dual with the primal. (Note that $\mathbb{Z}$GapSVP 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 with significantly larger $\lambda_1(\mathcal{L})$.)

Definition 4.6. *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 4.7. *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 2.4 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 4.8. *Let $r := \sqrt{\log n/(10\pi)}$, and let d be as in Lemma 4.1. Then, the above encryption scheme is correct, and for any $s = s(n) > 0$ and any $2^{-n} < \varepsilon < n^{-\omega(1)}$ the scheme is secure* either *if $(\alpha, \varepsilon, \rho)$-$\mathbb{Z}$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 $\gamma := 10\sqrt{n/\log(n/\varepsilon)} \cdot \alpha \approx \sqrt{10n/\log n}$.*

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

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

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

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

The relationship with $\mathbb{Z}$GapSPP is more interesting. In particular, there is a certain precise sense in which $\mathbb{Z}^n$ "has the largest smoothing parameter of any lattice." Such a statement can be made formal in a reverse Minkowski theorem [DR16, RS17]. In particular, recent work [RS17, RS22] comes quite close to proving a statement of the form "any lattice $\mathcal{L} \subset \mathbb{R}^n$ with determinant one and an integral Gram matrix $\boldsymbol{G}$ has $\eta_\varepsilon(\mathcal{L}) \leq \eta_\varepsilon(\mathbb{Z}^n)$, with equality if and only if $\mathcal{L} \cong \mathbb{Z}^n$."[8] Such a statement would suggest that (1) $\mathbb{Z}^n$ is the best lattice (among those with integral Gram matrices) for the purposes of decoding (since taking r above smoothing makes decoding impossible by definition); and (2) that solving $\mathbb{Z}$GapSPP might be more-or-less as hard as simply recognizing a rotation of $\mathbb{Z}^n$. In particular, it is trivial to check that the input lattice has an integral Gram matrix and determinant one, so recognizing $\mathbb{Z}^n$ is equivalent to distinguishing $\mathbb{Z}^n$ from any other such lattice. If all such lattices have smaller smoothing parameter than $\mathbb{Z}^n$, then distinguishing $\mathbb{Z}^n$ from a lattice with significantly smaller smoothing parameter is closely related to the problem of simply distinguishing $\mathbb{Z}^n$ from *any* lattice.

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

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

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

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

[8] Such lattices are called *unimodular*. Specifically, [RS22] prove that $\eta_\varepsilon(\mathcal{L}) \leq (2 + o(1))\eta_\varepsilon(\mathbb{Z}^n)$ for any unimodular lattice. Furthermore, [RS17] proves that $\eta_\varepsilon(\mathcal{L}) \leq \eta_\varepsilon(\mathbb{Z}^n)$ with equality if and only if $\mathcal{L} \cong \mathbb{Z}^n$ for very small $\varepsilon \leq 2^{-Cn}$, and [ERS22] proves a similar statement for a non-Gaussian test function.

5 Reductions and provable algorithms

In this section, we give a reduction from ℤSVP to *approximate* (unique-)SVP. In particular, our main result yields a randomized polynomial-time reduction from ℤSVP 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 ℤSVP.[9] This improves exponentially over the fastest known algorithm for SVP on general lattices [ADRS15], which runs in $2^{n+o(n)}$ time and was previously the fastest known algorithm even for the special case of ℤSVP. 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 [Ste16b] though it works in a very different regime (we solve *exact* ℤSVP using a γ-uSVP oracle for any constant γ, while [Ste16b] 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 ℤSVP, then it implies that there is no randomized polynomial-time algorithm for solving γ-uSVP for any constant $\gamma \geq 1$. This is notable because uSVP is not known to be NP-hard for any constant factor greater than 1. We also note that our main reduction generalizes to arbitrary lattices with few short vectors and may be of independent interest.

5.1 A simple projection-based reduction

Before giving our main reduction, we start with a simple reduction from ℤSVP to $\sqrt{2}$-SVP using a deterministic, "projection-based" approach. More specifically, we start by querying our $\sqrt{2}$-SVP oracle on the input lattice $\mathcal{L}$, and if the vector $\boldsymbol{v}$ that it returns does not have unit length then we recurse on the orthogonal projection $\pi_{\boldsymbol{v}^\perp}(\mathcal{L})$ of $\mathcal{L}$ onto $\boldsymbol{v}^\perp$. Although the results in Section 5.2 largely subsume it, we choose to present it because of its simplicity and the fact that the projection technique seems likely to generalize. (Indeed, we know similar reductions that work for $\sqrt{3}$-SVP and even 2-SVP, but our analysis of those involves substantial case analysis. We hope that there might be some more general reduction of which these are simply special cases, perhaps even a reduction that works for superconstant approximation factors.) It also has the advantage of only making at most two oracle queries.

Theorem 5.1. *There is a deterministic, dimension-preserving Cook reduction from* ℤSVP *to* $\sqrt{2}$-SVP.

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

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

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

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

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

5.2 The main reduction and algorithms

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

5.2.1 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 [ACK+21, Ste16a], but those results are in a slightly different form from what we need. However, because of its similarity to those results, we defer its proof to Appendix B.

Theorem 5.2. *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 5.2 holds for any $r > 0$, but that r need not be provided as input.

5.2.2 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 $G := N_{\mathrm{prim}}(\mathcal{L}, \lambda_1(\mathcal{L}))$.[10]

Theorem 5.3. *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 $(A/G) \cdot \mathrm{poly}(n)$ 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 [LLL82]), 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 5.2 with (a basis of) $\mathcal{L}$ and A' as input.
3. Return a shortest vector among the vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K$.

We start by proving correctness. Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_G$ be linearly independent shortest non-zero vectors in $\mathcal{L}$, i.e., let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_G \in \mathcal{L}$ be such that $\|\boldsymbol{x}_i\| = \lambda_1(\mathcal{L})$ and $\boldsymbol{x}_i \neq \pm \boldsymbol{x}_j$ for all $i \neq j$. (By assumption, G such vectors exist.) By Theorem 5.2 invoked with $r := \lambda_1(\mathcal{L})$, the probability that $\boldsymbol{y}_k = \boldsymbol{x}_i$ for any fixed $k \in [K]$ and $i \in [G]$ is at least $p := 1/(200A' \log(100A'))$. Moreover, because the events $\boldsymbol{y}_k = \boldsymbol{x}_i$ and $\boldsymbol{y}_k = \boldsymbol{x}_j$ are

[10] 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 $\mathcal{L}$.

disjoint $i \neq j$, the probability that there exists $i \in [G]$ such that $\boldsymbol{y}_k = \boldsymbol{x}_i$ for fixed $k \in [K]$ is at least Gp. So, the probability of some call to Theorem 5.2 returning a shortest non-zero vector $\boldsymbol{x}_i \in \mathcal{L}$ is at least

$$1 - (1 - Gp)^K \geq 1 - e^{-GpK} \geq 1 - e^{-n} \ ,$$

as needed.

We next turn to upper bounding the reduction's runtime. The reduction's overall runtime is dominated by the number of guesses needed for G' and A' in Step 1 times the runtime of the calls to the sampling algorithm in Theorem 5.2 made in Step 2.

Guessing G' and A' can be done by setting $G' := 2^{\ell}$ and $A' := 2^{\ell'}$ for integers ℓ, ℓ' satisfying $0 \leq \ell \leq \ell' \leq \log_2(2\gamma + 1) \cdot n$. Indeed, this suffices because $G' \leq A' \leq 2A$, and a straightforward packing argument shows that $A := N_{\mathrm{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L})) \leq (2\gamma + 1)^n$. Overall, Step 2 makes $K = O(A \log(A) n/G) \leq A/G \cdot \mathrm{poly}(n)$ calls to the sampling algorithm in Theorem 5.2, each of which makes a single call to a γ-uSVP oracle on a lattice of dimension n and uses $\mathrm{poly}(n)$ additional time. The reduction therefore makes at most

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

calls to its γ-uSVP oracle and uses $(A/G) \cdot \mathrm{poly}(n)$ time overall, as needed. □

5.2.3 Algorithms from Theorem 5.3

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 5.3, the point counting bound for $\mathbb{Z}^n$ in Proposition 2.12, the reduction from approximate uSVP to approximate GapSVP from Theorem 2.9, and the algorithm for $(1.93 + o(1))$-uSVP from Theorem 2.10 we get the following algorithmic result for $\mathbb{Z}$SVP.

Corollary 5.4. *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 of dimension n in $2^{n/2+o(n)}$ time.*

Proof. By the rotational invariance of the ℓ_2 norm and Proposition 2.12,

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

The main result then follows immediately from Theorem 5.3.

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 2.10 with the efficient dimension-preserving reduction from uSVP to GapSVP in Theorem 2.9. □

We again emphasize that the $2^{n/2+o(n)}$-time algorithm in Corollary 5.4 substantially improves over the $2^{n+o(n)}$-time SVP algorithm for general lattices from [ADRS15], which was also the previous fastest known algorithm for $\mathbb{Z}$SVP.

In fact, Theorem 5.3 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 5.3 with the point-counting bound for semi-stable lattices in Proposition 2.13 gives such an algorithm.

Corollary 5.5. *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 2.13 to upper bound $A := N_{\mathrm{prim}}(\mathcal{L}, \gamma \cdot \lambda_1(\mathcal{L}))$ and then invoking Theorem 5.3. 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 because rotations $\mathcal{L}$ of $\mathbb{Z}^n$ are semi-stable lattices with $\lambda_1(\mathcal{L}) = 1$, Corollary 5.5 implies a $2^{n/2+o(n)}$-time algorithm for $\mathbb{Z}$SVP and hence subsumes the "in particular" part of Corollary 5.4. However, Corollary 5.4 says something somewhat stronger for $\mathbb{Z}$SVP than Corollary 5.5. Namely, Corollary 5.4 shows how to turn a $2^{cn+o(n)}$-time algorithm for γ-uSVP with any $\gamma = o(\sqrt{n})$ into a $2^{cn+o(n)}$-time algorithm for $\mathbb{Z}$SVP, whereas getting this result from Corollary 5.5 requires an algorithm for γ-uSVP with $\gamma = o(\sqrt{n}/\log n)$.

We also note that Theorem 5.3 and Corollaries 5.4 and 5.5 answer a special case of an interesting question of Ducas and van Woerden [DvW22], 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).

5.2.4 Hardness from Theorem 5.3

Corollaries 5.4 and 5.5 combine the reduction in Theorem 5.3 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 $\gamma = 1 + 1/\operatorname{poly}(n)$; see [AD16, Ste16b]. 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 $\operatorname{dist}(\boldsymbol{t}, \mathcal{L}) \leq \alpha \cdot \lambda_1(\mathcal{L})$ as input, the goal is to output a closest lattice point to $\boldsymbol{t}$ (i.e., $\boldsymbol{x} \in \mathcal{L}$ satisfying $\|\boldsymbol{t} - \boldsymbol{x}\| = \operatorname{dist}(\boldsymbol{t}, \mathcal{L})$).

Corollary 5.6. *The following hardness results hold for* γ-uSVP *and* α-BDD*:*

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

Proof. The contrapositive of the claims for uSVP follow immediately from Corollaries 5.4 and 5.5. The claims for BDD follow from this by additionally noting that [LM09] gives an efficient reduction from γ-uSVP to $(1/\gamma)$-BDD for any $\gamma = \gamma(n) \leq \operatorname{poly}(n)$. □

6 Experiments

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

6.1 Experiments on different procedures for generating bases

In this section, we present experimental results examining the effectiveness of standard basis reduction algorithms for solving $\mathbb{Z}$SVP. Specifically, we generate bases of $\mathbb{Z}^n$ (which we then treat as instances of $\mathbb{Z}$SVP) using three procedures: discrete-Gaussian-based sampling, unimodular-matrix-product-based sampling, and

Bézout-coefficient-based sampling. Using each of these procedures, we generate bases in dimensions $n = 128$, 256, and 512 with a variety of settings for procedure-specific parameters.[11] These results extend those in [BM21], which included experiments on bases generated using the second two procedures.

For each basis generating procedure (and corresponding set of parameters), we run the LLL algorithm and BKZ reduction algorithm (as implemented in fplll [FPL]) with different block sizes. For BKZ, we use block sizes 3, 4, 5, 10, and 20—though in dimension 512, we left out block size 20 for most of our experiments due to computational constraints. We 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 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, such as the running times of our experiments, the squared length of the shortest vector found in each trial, and the code used, can be found in the associated repository [BGPS21].

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

We note that the "reasonable parameter" regime for the discrete Gausian-based generating procedure yields significantly shorter vectors than the other procedures, which might be considered an advantage, as it means that the resulting bases can be stored more efficiently.

6.1.1 Discrete Gaussian-based sampling.

We start by presenting the results of experiments performed on bases generated essentially as described in Section 3 (which is also what we use for our encryption scheme in Section 4). However, we make three minor modifications. First, instead of sampling vectors one at a time until we find a generating set of $\mathbb{Z}^n$, we simply sample $n + 10$ vectors. (There is nothing special about the number 10 here.) Empirically, we found that this yielded a generating set with high probability. Notice that this is much better than what is proven in Lemma 2.5. See also [NP22].

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 11).

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, and we include data with $s = 1$ only for completeness. Nevertheless, interestingly, in all twenty runs, we were actually unable to find a shortest vector even for $s = 1$ in dimension $n = 512$.

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

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

Discrete Gaussian-based sampling

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

Algorithm:

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

6.1.2 Unimodular matrix product sampling.

The second basis sampling technique that we analyze was proposed in [BM21], where it is called Algorithm 3.[12] 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.

Basis Generating Procedure 2: Unimodular matrix product

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

Algorithm:

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

In our experiments, we considered all combinations of parameters $d \in \{2, 3, 4\}$, $B = 1$, and $L \in \{10n, 20n, 30n, 40n, 50n\}$, except that we did 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 [BM21].)

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

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

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

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

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

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

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

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

6.1.3 Bézout-coefficient-based sampling.

We next describe our third basis-sampling algorithm, which was suggested by Joseph Silverman and studied as Algorithm 4 in [BM21]. 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 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 [BM21, Algorithm 4].)

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

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

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

Basis Generating Technique 3: Bézout-coefficient-based construction

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

Algorithm:

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

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

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

6.2 An interesting threshold phenomenon

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

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

				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 6.1.2. 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.

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

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

get more fine-grained results than those from Section 6.1, we ran LLL and then sequentially ran BKZ with all block sizes between 3 and 10 (rather than skipping block sizes 6, 7, 8, and 9). For these experiments, we were not particularly interested in the block size that was needed to find a shortest non-zero vector.[13] Instead, we were interested in the length of the shortest vector found by a run of BKZ when it did not find a vector with length one.

The results are striking. See Figure 1 and Figure 2. In particular, in 1000 experiments, our algorithms never returned a basis whose shortest vector had length between $\sqrt{3}$ and $\sqrt{28}$. (Strangely, there are four examples when the shortest vector in the basis had length $\sqrt{2}$. We have no explanation for this.) And, e.g., there are only fifteen examples in which this length was less than $\sqrt{35}$ but not equal to one.

To our knowledge, a similar phenomenon is not known for other lattices.[14] We think it deserves further study.

We have no truly rigorous explanation for this. But, we can offer some vague and speculative intuition for what is going on. To that end, we observe that vectors with length less than roughly $\sqrt{n}/2$ in $\mathbb{Z}^n$ are orthogonal to many shortest non-zero vectors in $\mathbb{Z}^n$. Since basis reduction algorithms work with the lattice projected orthogonally to the basis vectors, we suspect that this offers a significant advantage to the algorithm, which might be what is reflected in the data.

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

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 $\mathbb{Z}^n$ violates 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

[13] We found a shortest non-zero vector eventually in all experiments in dimension $n = 128$, and only 8 of the 1000 trials required block size 10 to find a shortest non-zero vector. See [BGPS21] for the raw data.

[14] There are examples of lattices for which basis reduction exhibits a threshold phenomenon that is superficially similar. E.g., lattices with $\lambda_1(\mathcal{L}) \ll \lambda_2(\mathcal{L})$ naturally exhibit such a threshold at length $\lambda_2(\mathcal{L})$, but this is not surprising and seems like a qualitatively different phenomenon. More generally, lattices with unusually dense subspaces, such as NTRU lattices exhibit a threshold [Wal21], but this again seems like a qualitatively different phenomenon.

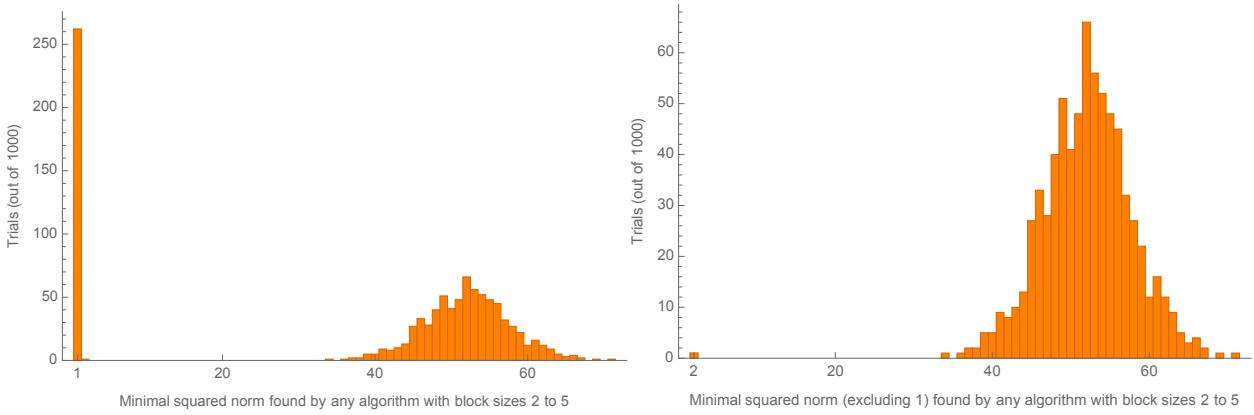

Figure 1: On the left is a histogram of the squared norm of the shortest vector found by BKZ with block size ≤ 5 for discrete Guassian bases with $n = 128$ and $s = 1000$. On the right is the same histogram without the trials where this norm was 1.

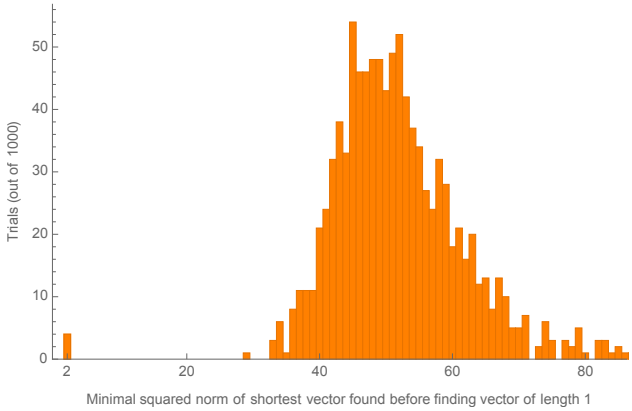

Figure 2: A histogram of the minimal squared length of a basis vector found before finding a vector with length one after 1000 trials on discrete Guassian bases with $n = 128$ and $s = 1000$. E.g., if in one experiment BKZ with block size 6 finds a vector with length one, but the shortest vector found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

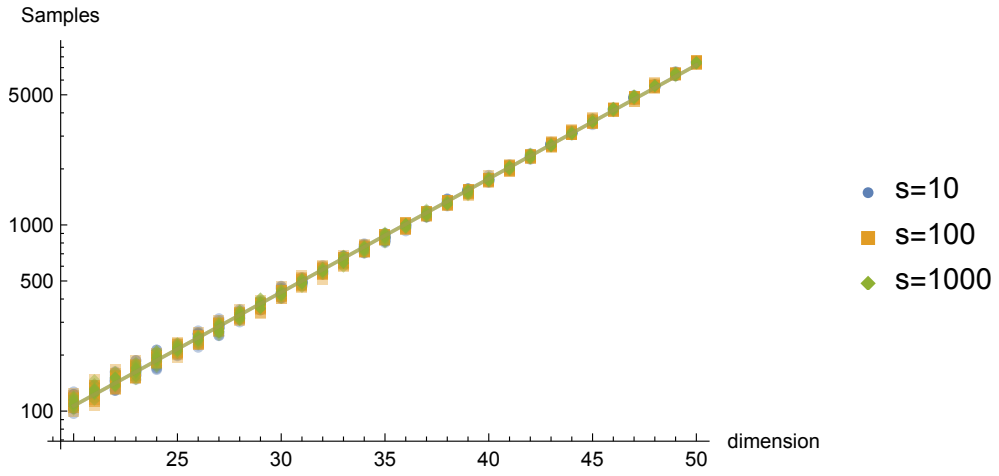

Figure 3: Scatter plot of the number of vectors sampled by the sieving algorithm in different dimensions with different parameters s, together with the fitted line $6.4 \cdot 1.15^n$. (The fact that the three different parameter values are not distinguishable in the plot reflects the fact that the number of sampled vectors was essentially independent of the parameter size, which is to be expected.)

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 2.12), 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 [MV10], 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 essentially random from the set of such vectors. This heuristic matches the data reasonably well, though failure is slightly more common than this simple heuristic suggests, and the difference is on the verge of statistically significant. (If we instead wait until we find, e.g., 10 collisions, the failure rate becomes extremely low.)

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 3. 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 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. See Figure 4. Of course, this running time is subject to many minor implementation details. A less fickle

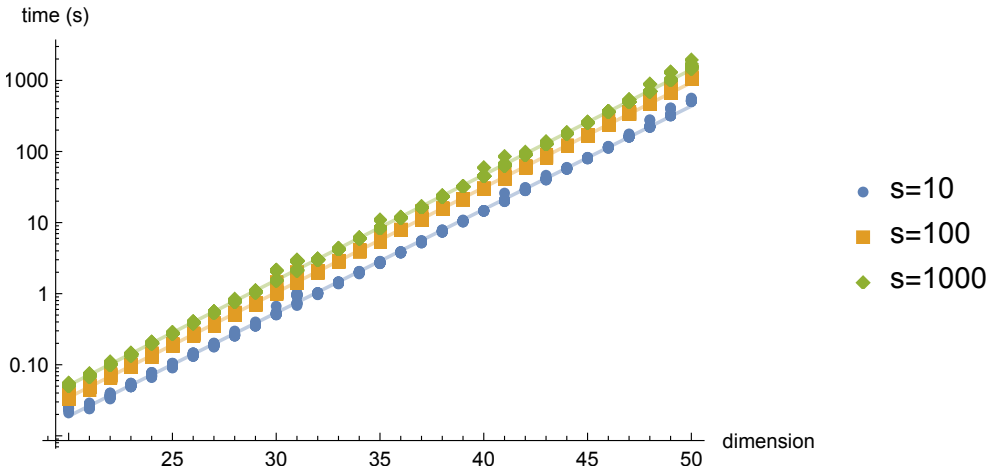

Figure 4: The running time of our implementation of the Gauss sieve algorithm on $\mathbb{Z}^n$ with different Gaussian parameters s. The trend lines are (roughly) $1.4^n/43000$ seconds, $1.41^n/26000$ seconds, and $1.41^n/18000$ seconds respectively.

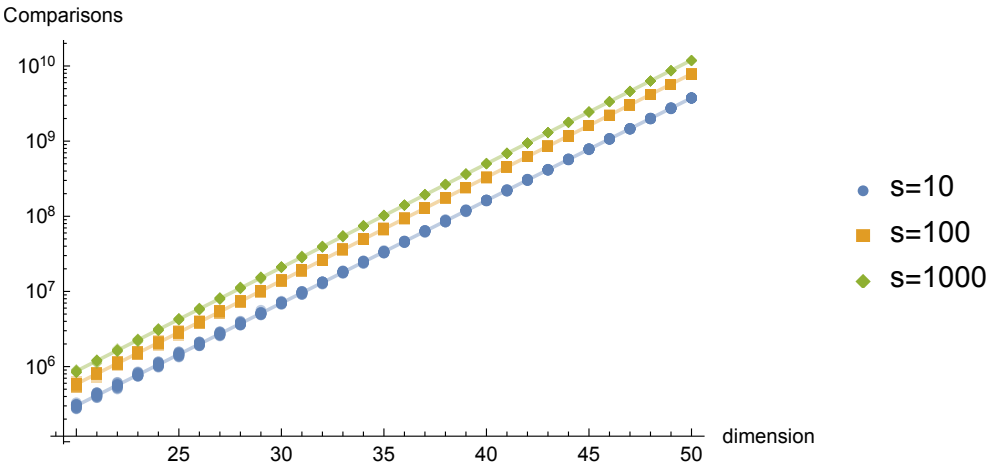

Figure 5: The number of comparisons made by Micciancio and Voulgaris's Gauss sieve algorithm on $\mathbb{Z}^n$ with different Gaussian parameters s. The trend lines are (roughly) $500 \cdot 1.37^n$, $1000 \cdot 1.37^n$, and $1500 \cdot 1.37^n$ respectively.

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 5. 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 [BGPS21].

References

[ACK+21] Divesh Aggarwal, Yanlin Chen, Rajendra Kumar, Zeyong Li, and Noah Stephens-Davidowitz. Dimension-preserving reductions between SVP and CVP in different p-norms. In *SODA*, 2021. 19

[AD16] Divesh Aggarwal and Chandan K. Dubey. Improved hardness results for unique shortest vector problem. *Inf. Process. Lett.*, 116(10):631–637, 2016. 3, 21

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete Gaussian sampling. In *STOC*, 2015. 1, 2, 6, 8, 18, 20

[AEN] Yoshinoro Aono, Thomas Espitau, and Phong Q. Nguyen. Random lattices: theory and practice. https://espitau.github.io/bin/random_lattice.pdf. i, 4, 5, 10

[BGPS21] Huck Bennett, Atul Ganju, Pura Peetathawatchai, and Noah Stephens-Davidowitz. Experiments on solving SVP on rotations of $\mathbb{Z}^n$. https://github.com/poonpura/Experiments-on-Solving-SVP-on-Rotations-of-Z-n, 2021. 21, 22, 27, 31

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

[BM21] Tamar Lichter Blanks and Stephen D. Miller. Generating cryptographically-strong random lattice bases and recognizing rotations of $\mathbb{Z}^n$. In *PQCrypto*, 2021. i, 4, 5, 6, 22, 23, 24, 25

[BSW16] Shi Bai, Damien Stehlé, and Weiqiang Wen. Improved reduction from the Bounded Distance Decoding problem to the unique Shortest Vector Problem in lattices. In *ICALP*, 2016. 3

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

[CGG17] Karthekeyan Chandrasekaran, Venkata Gandikota, and Elena Grigorescu. Deciding orthogonality in Construction-A lattices. *SIAM Journal on Discrete Mathematics*, 31(2):1244–1262, January 2017. 1, 5

[CHKP12] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *J. Cryptol.*, 25(4):601–639, 2012. Preliminary version in EUROCRYPT 2010. i, 4, 5, 10

[DR16] Daniel Dadush and Oded Regev. Towards strong reverse Minkowski-type inequalities for lattices. In *FOCS*, 2016. 3, 17

[DvW22] Léo Ducas and Wessel van Woerden. On the lattice isomorphism problem, quadratic forms, remarkable lattices, and cryptography. In *EUROCRYPT*, 2022. i, 2, 4, 5, 10, 21

[ERS22] Yael Eisenberg, Oded Regev, and Noah Stephens-Davidowitz. A tight reverse Minkowski inequality for the Epstein zeta function. arXiv:2201.05201, January 2022. 3, 17

[FPL] FPLLL development team. fplll, a lattice reduction library, Version: 5.4.1. Available at https://github.com/fplll/fplll. 22

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008. 4, 7

[GS02] Craig Gentry and Michael Szydlo. Cryptanalysis of the revised NTRU signature scheme. In *EUROCRYPT*, 2002. 1, 5

[GS03] Katharina Geißler and Nigel P. Smart. Computing the $M = U^T U$ integer matrix decomposition. In *Cryptography and Coding*, 2003. 1, 5

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

[HR14] Ishay Haviv and Oded Regev. On the Lattice Isomorphism Problem. In *SODA*, 2014. i, 4, 5, 8, 10

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

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

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982. 19

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem. In *CRYPTO*, 2009. 3, 8, 21

[LS14] H. W. Lenstra and A. Silverberg. Revisiting the Gentry-Szydlo algorithm. In *CRYPTO*, 2014. 1, 5

[LS17] H. W. Lenstra and A. Silverberg. Lattices with symmetry. *Journal of Cryptology*, 30(3):760–804, 2017. 1, 5

[MH73] John Willard Milnor and Dale Husemöller. *Symmetric Bilinear Forms*. Springer-Verlag, 1973. 17

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

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal of Computing*, 37(1):267–302, 2007. 8

[MV10] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the Shortest Vector Problem. In *SODA*, 2010. 4, 29

[NP22] Phong Q. Nguyen and Löic Pujet. The probability of primitive sets and generators in lattices, 2022. 6, 22

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

[Reg04] Oded Regev. LLL algorithm. https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/lll.pdf, 2004. 12

[RS17] Oded Regev and Noah Stephens-Davidowitz. A reverse Minkowski theorem. In *STOC*, 2017. 3, 9, 17

[RS22] Oded Regev and Noah Stephens-Davidowitz. A reverse Minkowski theorem for integral lattices. 2022. 3, 17

[Ste16a] Noah Stephens-Davidowitz. Discrete Gaussian sampling reduces to CVP and SVP. In *SODA*, 2016. 19, 34

[Ste16b] Noah Stephens-Davidowitz. Search-to-decision reductions for lattice problems with approximation factors (slightly) greater than one. In *APPROX*, 2016. 3, 5, 18, 21

[Ste20] Noah Stephens-Davidowitz. Lattice algorithms. https://www.youtube.com/watch?v=o4Pl-0Q5-q0, 2020. Talk as part of the Simons Institute's semester on lattices. 2, 18

[Szy03] Michael Szydlo. Hypercubic lattice reduction and analysis of GGH and NTRU signatures. In *EUROCRYPT*, 2003. 1, 5, 27

[Wal21] Michael Walter. personal communication, 2021. 27

A Proof of Lemma 2.1

We first study the expectation of $(X \bmod \mathbb{Z})^2$ for $X \sim D_s$. We then use the Chernoff-Hoeffding (Lemma 2.14) bound to obtain the result. For $t \in (-1/2, 1/2]$, let $p(t) := (1/s) \cdot \sum_{z \in \mathbb{Z}} \rho_s(t+z)$ be the probability density function of $X \bmod \mathbb{Z} \in (-1/2, 1/2]$. By the Poisson summation formula, we have

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

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

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

Notice that

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

Therefore,

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

for some χ with

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

By the Chernoff-Hoeffding bound (Lemma 2.14),

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

It follows that

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

as needed.

B Proof of Theorem 5.2

We next state a key result on lattice *sparsification* that we will use to prove Theorem 5.2. Roughly speaking, it implies that if we sample a random sublattice $\mathcal{L}'$ of prime index p of a lattice $\mathcal{L}$, and $1 < N_{\text{prim}}(\mathcal{L}, r) \lesssim p/\log p$ then $\mathcal{L}'$ contains a given primitive lattice vector $\boldsymbol{x}_0 \in \mathcal{L}$ and no other primitive vectors in $\mathcal{L}$ with probability around $1/p$.

Theorem B.1 ([Ste16a, Theorem 4.1])**.** *For any lattice $\mathcal{L} \subseteq \mathbb{R}^n$ with basis B, $r > 0$, primitive lattice vectors $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathcal{L}_{prim}$ satisfying $\|\boldsymbol{x}_i\| \leq r$ for all i and $\boldsymbol{x}_i \neq \pm\boldsymbol{x}_0$ for all $i > 0$, and prime $p \geq 101$, if $N_{prim}(\mathcal{L}, r) \leq p/(20 \log p)$ then*

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

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

We now restate and prove Theorem 5.2.

Theorem 5.2. *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 $\mathcal{L}$.

Proof. We assume without loss of generality that $r \geq \lambda_1(\mathcal{L})$. Otherwise, the claim is vacuous because there are no primitive lattice vectors of norm less than $\lambda_1(\mathcal{L})$, i.e., $\mathbf{0}$ is not primitive. Additionally, we assume without loss of generality that the γ-uSVP oracle always returns a lattice vector even when the lattice generated by its input basis does not satisfy γ-uSVP promise. Let $\mathbf{B}$ be the input basis of $\mathcal{L}$, and let n be the rank of $\mathcal{L}$.

The algorithm first computes a prime p satisfying $50A' \log(100A') \leq p \leq 100A' \log(100A')$. It then samples $\boldsymbol{a} \sim \mathbb{Z}_p^n$ uniformly at random and computes a basis $\mathbf{B}'$ of the sublattice

$$\mathcal{L}' := \{\boldsymbol{v} \in \mathcal{L} : \langle \mathbf{B}^{-1}\boldsymbol{v}, \boldsymbol{a} \rangle \equiv 0 \pmod p\} .$$

Next, it calls the γ-uSVP oracle on $\mathbf{B}'$, and receives as output a vector $\boldsymbol{y} \in \mathcal{L}$. Finally, it returns $\boldsymbol{y}$.

Computing a valid prime p can be done efficiently by repeatedly sampling a uniformly random integer in the range $[50A' \log(100A'), 100A' \log(100A')]$ and testing whether it is prime. Additionally, $\mathbf{B}'$ can be computed efficiently given $\mathbf{B}$, p, and $\boldsymbol{a}$ (see, e.g., [Ste16a, Claim 2.15]). So, the algorithm runs in polynomial time and makes a single call to its γ-uSVP oracle, as claimed.

We next show correctness. Let $\boldsymbol{x}_0 := \boldsymbol{x}$, and let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{A-1}$ be primitive vectors in $\mathcal{L}$ satisfying $\boldsymbol{x}_i \neq \pm\boldsymbol{x}_j$ for all $0 \leq i < j \leq A-1$. Define the event

$$E_0 := [\boldsymbol{x}_0 \in \mathcal{L}' \text{ and } \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{A-1} \notin \mathcal{L}'] .$$

Because $N_{\text{prim}}(\mathcal{L}, \gamma r) = A$, if E_0 holds then $\boldsymbol{x}_0$ is the unique (up to sign) primitive vector in $\mathcal{L}'$ of norm at most γr. So, if E_0 holds, $\mathcal{L}'$ meets the γ-uSVP promise, and the γ-uSVP oracle must return $\boldsymbol{y} = \boldsymbol{x} = \boldsymbol{x}_0$.

We will use Theorem B.1 to lower bound $\Pr[E_0]$, but in order to apply it we need to show that $A' \leq p/(20 \log p)$. Using the fact that $50A' \log(100A') \leq p \leq 100A' \log(100A')$, for this to hold it suffices to have

$$\frac{50A' \log(100A')}{20 \log(100A' \log(100A'))} \geq A' ,$$

and this holds because $20 \log(100A' \log(100A') \leq 40 \log(100A')$. So, by Theorem B.1 we have

$$\Pr[E_0] \geq \frac{1}{p} - \frac{A-1}{p^2} \geq \frac{1}{p} - \frac{A'}{p^2} \geq \frac{1}{p} - \frac{1}{50 \log(100A')p} \geq \frac{1}{2p} \geq \frac{1}{200A' \log(100A')} ,$$

as needed. □