Solving Hard Lattice Problems and the Security of Lattice-Based Cryptosystems

Thijs Laarhoven* Joop van de Pol† Benne de Weger*

September 10, 2012

Abstract

This paper is a tutorial introduction to the present state-of-the-art in the field of security of lattice-based cryptosystems. After a short introduction to lattices, we describe the main hard problems in lattice theory that cryptosystems base their security on, and we present the main methods of attacking these hard problems, based on lattice basis reduction. We show how to find shortest vectors in lattices, which can be used to improve basis reduction algorithms. Finally we give a framework for assessing the security of cryptosystems based on these hard problems.

1 Introduction

Lattice-based cryptography is a quickly expanding field. The last two decades have seen exciting developments, both in using lattices in cryptanalysis and in building public key cryptosystems on hard lattice problems. In the latter category we could mention the systems of Ajtai and Dwork [AD], Goldreich, Goldwasser and Halevi [GGH2], the NTRU cryptosystem [HPS], and systems built on Small Integer Solutions [MR1] problems and Learning With Errors [R1] problems. In the last few years these developments culminated in the advent of the first fully homomorphic encryption scheme by Gentry [Ge].

A major issue in this field is to base key length recommendations on a firm understanding of the hardness of the underlying lattice problems. The general feeling among the experts is that the present understanding is not yet on the same level as the understanding of, e.g., factoring or discrete logarithms. So a lot of work on lattices is needed to improve this situation.

This paper aims to present a tutorial introduction to this area. After a brief introduction to lattices and related terminology in Section 2, this paper will cover the following four main topics:

- Section 3: Hard Lattice Problems,
- Section 4: Solving the Approximate Shortest Vector Problem,
- Section 5: Solving the Exact Shortest Vector Problem,
- Section 6: Measuring the Practical Security of Lattice-Based Cryptosystems.

Section 3 on hard lattice problems presents a concise overview of the main hard lattice problems and their interrelations. From this it will become clear that the so-called *Approximate Shortest Vector Problem* is the central one. The paramount techniques for solving these problems are the so-called lattice basis

*Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands.
†Department of Computer Science, University of Bristol, United Kingdom.

reduction methods. In this area a major breakthrough occurred in 1982 with the development of the LLL algorithm [LLL]. An overview of these and later developments will be given in Section 4, with a focus on practical algorithms and their performance. In Section 5, techniques for solving the *Exact Shortest Vector Problem*, such as enumeration and sieving, are discussed. These techniques can be used to find shortest vectors in low dimensions, and to improve lattice basis reduction algorithms in high dimensions. Finally, in Section 6, we look at lattice-based cryptography. Using theoretical and experimental results on the performance of the lattice basis reduction algorithms, we aim to get an understanding of the practical security of lattice-based cryptosystems.

For evaluating the practicality of secure cryptosystems, both security issues and issues of performance and complexity are of paramount importance. This paper is only about the security of lattice based cryptosystems. Performance and complexity of such cryptosystems are out of scope for this paper.

2 Lattices

This section gives a brief introduction to lattices, aiming at providing intuition rather than precision. A much more detailed recent account can be found in Nguyen's paper [N], on which this section is loosely based.

2.1 Lattices

The theory of lattices may be described as *discrete linear algebra*. The prime example for the concept of lattice is $\mathbb{Z}^n$, the set of points in the n-dimensional real linear space $\mathbb{R}^n$ with all coordinates being integers. This set forms a group under coordinate-wise addition, and is discrete, meaning that for every point in the set there is an open ball around it that contains no other point of the set. On applying any linear transformation to $\mathbb{Z}^n$ the additive group structure and the discreteness are preserved. Any such image of $\mathbb{Z}^n$ under a linear transformation is called a *lattice*. Indeed, any discrete additive subgroup of $\mathbb{R}^n$ can be obtained as a linear transformation of $\mathbb{Z}^n$. Therefore in the literature lattices are often defined as *discrete additive subgroups of* $\mathbb{R}^n$.

In this paper we further restrict to lattices in $\mathbb{Z}^n$. The linear transformation applied to $\mathbb{Z}^n$ that produces the lattice transforms the standard basis of $\mathbb{Z}^n$ into a set of *generators* for the lattice, meaning that the lattice is just the set of *integral linear combinations* of those generators (i.e. linear combinations with integer coefficients). The set of generators can be reduced to a linearly independent set of generators, called a *basis*. Following this line of thought, in this paper we adopt a more down-to-earth definition of lattices.

Definition 2.1. A *lattice* is a set of all integral linear combinations of a given set of linearly independent points in $\mathbb{Z}^n$. For a basis $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ we denote the lattice it generates by

$$L(\mathbf{B}) = \left\{ \sum_{i=1}^{d} x_i \mathbf{b}_i \mid x_i \in \mathbb{Z} \right\}.$$

Its *rank* is d, and the lattice is said to be of *full rank* if $d = n$. We identify the basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ with the $n \times d$ matrix containing $\mathbf{b}_1, \ldots, \mathbf{b}_d$ as columns, which enables us to write the shorter $L(\mathbf{B}) = \{\mathbf{Bx} \mid \mathbf{x} \in \mathbb{Z}^d\}$.

We use both the terms *lattice point* and *lattice vector* for elements of a lattice.

We proceed with a few trivial facts. Obviously, a lattice has many different bases (except in the uninteresting case of rank

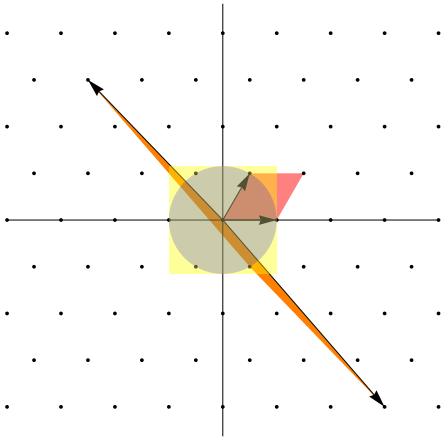

Figure 1: A lattice with two different bases and their fundamental domains, with unit balls for the Euclidean and supremum norms.

1). Indeed, $L(\mathbf{B}) = L(\mathbf{BU})$ for any $d \times d$ integral matrix $\mathbf{U}$ with $\det(\mathbf{U}) = \pm 1$, and any basis of $L(\mathbf{B})$ is of this form $\mathbf{BU}$. See Figure 1 for a lattice with two different bases.

Associated to a basis $\mathbf{B}$ is its *fundamental domain* in $\mathbb{R}^n$, given as $\{\mathbf{Bx} \mid \mathbf{x} \in [0,1)^d\}$. Any point in the real span of $\mathbf{B}$ can be decomposed uniquely into the sum of a lattice point and a point in the fundamental domain. The *volume* of a lattice of rank d is the d-dimensional volume of the fundamental domain of a basis. Figure 1 shows in orange shading the fundamental domains for the two bases. It follows that $\mathrm{vol}(L(\mathbf{B})) = \sqrt{\det(\mathbf{B}^\top \mathbf{B})}$, and in the full rank case $\mathrm{vol}(L(\mathbf{B})) = |\det(\mathbf{B})|$. The volume is independent of the choice of basis, since $\det((\mathbf{BU})^\top \mathbf{BU}) = \det(\mathbf{U}^\top \mathbf{B}^\top \mathbf{BU}) = \det(\mathbf{U}^\top)\det(\mathbf{B}^\top \mathbf{B})\det(\mathbf{U}) = \det(\mathbf{B}^\top \mathbf{B})$ by $\det(U) = \pm 1$, so for any lattice L the volume $\mathrm{vol}(L)$ is well-defined.

2.2 Norms

Many lattice problems are about *distances*. The distance between two points is defined as the *norm* of their difference: $d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|$, where a *norm* (also called *length*) on $\mathbb{R}^n$ is a function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ satisfying

- $\|\mathbf{x}\| > 0$ for all $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, and $\|\mathbf{0}\| = 0$,
- $\|t\mathbf{x}\| = |t|\|\mathbf{x}\|$ for all $\mathbf{x} \in \mathbb{R}^n, t \in \mathbb{R}$,
- $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ for all $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$.

The main example is the Euclidean norm $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^\top \mathbf{x}}$, but many other examples exist, such as the supremum norm $\|\mathbf{x}\|_\infty = \max_i |x_i|$ for $\mathbf{x} = (x_1,\ldots,x_n)$, the p-norm $\|\mathbf{x}\|_p = (\sum_i |x_i|^p)^{1/p}$ for any $p \geq 1$, and norms coming from inner products. Figure 1 shows unit balls for two norms. Note that for any two norms $\|\cdot\|_\alpha, \|\cdot\|_\beta$ there exists an optimal constant $N_{\beta,\alpha}$ such that $\|\cdot\|_\beta \leq N_{\beta,\alpha}\|\cdot\|_\alpha$. Unfortunately this constant in general depends on the dimension, e.g. $N_{2,\infty} = \sqrt{n}$, but $N_{\infty,2} = 1$. In particular, for any norm $\|\cdot\|_*$ we define $N_* = N_{\infty,*}$. For a matrix $\mathbf{M}$ with columns $\mathbf{m_i}$ we define a norm as $\|\mathbf{M}\|_* = \max_i \|\mathbf{m_i}\|_*$. It follows that $\|\mathbf{Mx}\|_* \leq \|\mathbf{M}\|_*\|\mathbf{x}\|_\infty \leq N_*\|\mathbf{M}\|_*\|\mathbf{x}\|_*$.

The theory of lattice problems and lattice algorithms can be set up for general norms. Let us give a naive example, with a naive solution, just to see what sort of problems will come up. For a given norm $\|\cdot\|_*$ (that is supposed to be defined on both $\mathbb{R}^d$ and $\mathbb{R}^n$) and any lattice L, one problem that can be studied is to determine all lattice points inside a given ball centered around the origin, i.e. to find all $\mathbf{y} \in L$ with $\|\mathbf{y}\|_* \leq r$, for some radius $r > 0$. The so-called *Gaussian Heuristic* [N, Definition 8] says that the number of such points is approximately $v_{d*} r^d / \mathrm{vol}(L)$, where v_{d*} is the volume of the d-dimensional unit ball for the norm $\|\cdot\|_*$. It follows that in order to have any chance of finding a nonzero solution, r must be at least of the size of $\mathrm{vol}(L)^{1/d}$. For the Euclidean norm it holds [N, Corollary 3] that there is a nonzero solution when $r \geq \sqrt{d} \cdot \mathrm{vol}(L)^{1/d}$. But it may be hard to find it in practice.

When a basis $\mathbf{B}$ of the lattice is given, this problem becomes finding all $\mathbf{x} \in \mathbb{Z}^d$ such that $\|\mathbf{Bx}\|_* \leq r$. Now note that by definition $\mathbf{B}^\top \mathbf{B}$ is invertible. So from $\mathbf{y} = \mathbf{Bx}$ it follows that $\mathbf{x} = \left(\mathbf{B}^\top \mathbf{B}\right)^{-1} \mathbf{B}^\top \mathbf{y}$, and thus from $\|\mathbf{y}\|_* \leq r$ it follows that, with $F_*(\mathbf{B}) = N_* \| \left(\mathbf{B}^\top \mathbf{B}\right)^{-1} \mathbf{B}^\top \|_*$, it suffices to search for all $\mathbf{x} \in \mathbb{Z}^d$ with $\|\mathbf{x}\|_* \leq F_*(\mathbf{B}) r$. Brute force techniques then can be invoked to compile a list of all these $\mathbf{x}$, and each of them can then be further tested for $\|\mathbf{Bx}\|_* \leq r$. Note that for full rank lattices, $\mathbf{B}$ itself is invertible, so then $\left(\mathbf{B}^\top \mathbf{B}\right)^{-1} \mathbf{B}^\top = \mathbf{B}^{-1}$, and $F_*(\mathbf{B}) = N_* \|\mathbf{B}^{-1}\|_*$.

The complexity of this method will be proportional to $(F_*(\mathbf{B}) r)^d$. The Gaussian Heuristic says that this is reasonable when it is approximately $r^d / \mathrm{vol}(L)$ (ignoring minor terms), so that we want $F_*(\mathbf{B}) \approx \mathrm{vol}(L)^{-1/d}$. The main problem here is that the multiplication factor $F_*(\mathbf{B})$ depends heavily on the basis and the dimension.

2.3 Good and bad bases

Assume, for simplicity, that we have a norm $\|\cdot\|_*$ that comes from an inner product, and that we have a basis of full rank (so $d = n$) that is orthonormal with respect to this norm. Then clearly $\|\mathbf{B}^{-1}\|_* = 1$, so that $F_*(\mathbf{B}) = N_*$ which is probably not too big, and is anyway independent of the basis. This is a typical example of a 'good' basis. A similar argument can be given for 'orthogonal' bases, by scaling. Note that for any basis $\mathrm{vol}(L) \leq \prod_i \|\mathbf{b}_i\|_*$, with equality just for orthogonal bases. In general we can say that a basis is 'good' if the number $\prod_i \|\mathbf{b}_i\|_* / \mathrm{vol}(L)$ is small (i.e. not much larger than 1). On assuming that all basis points have approximately equal norms, this condition becomes even easier, namely that $\|\mathbf{B}\|_*^n / \mathrm{vol}(L)$ is not much bigger than 1. In such a case Cramer's rule tells us that $\|\mathbf{B}^{-1}\|_* \approx \|\mathbf{B}\|_*^{n-1} / \mathrm{vol}(L) \approx \|\mathbf{B}\|_*^{-1} \approx \mathrm{vol}(L)^{-1/n}$, so that the multiplication factor $F_*(\mathbf{B})$ indeed has the required size $\mathrm{vol}(L)^{-1/n}$, as expected from the Gaussian Heuristic.

But if we have a basis that is far from orthogonal, in the sense that $\|\mathbf{B}\|_*^n / \mathrm{vol}(L)$ is large, say $\mathrm{vol}(L)^\alpha$ for some $\alpha > 0$, then the multiplication factor $F_*(\mathbf{B})$ will be of the size of $\mathrm{vol}(L)^{\alpha(1-1/n)-1/n}$, and so with $r \approx \mathrm{vol}(L)^{1/n}$ the complexity of the brute force technique will be $\mathrm{vol}(L)^{\alpha(n-1)}$, which is really bad.

As a typical example, look at $\mathbf{B}_\varepsilon = \begin{pmatrix} p & \lambda p \\ q & \lambda q + \varepsilon \end{pmatrix}$ for a very small ε, which has $\det(\mathbf{B}_\varepsilon) = p\varepsilon$, almost dependent columns, so is 'far from good', and $\|\mathbf{B}_\varepsilon\|_*$ almost independent of ε, while $\mathbf{B}_\varepsilon^{-1} = \varepsilon^{-1} \begin{pmatrix} (\lambda q + \varepsilon)/p & -\lambda \\ -q/p & 1 \end{pmatrix}$, leading to $\|\mathbf{B}_\varepsilon^{-1}\|_*$ growing almost proportional to ε^{-1}.

This simple heuristic argument shows that it will be very useful to have bases as 'good' as possible, i.e. as 'orthogonal' as possible. But this depends on the norm. We will now show that for every basis a 'good' norm can be found easily.

Indeed, to a (full rank, for convenience) lattice with basis $\mathbf{B}$ the inner product $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{B}} = \mathbf{x}^\top \mathbf{B}^{-1\top} \mathbf{B}^{-1} \mathbf{y}$ can be associated. This inner product defines the associated norm as the square root of a positive definite quadratic form: $\|\mathbf{x}\|_{\mathbf{B}} = \sqrt{\mathbf{x}^\top \mathbf{B}^{-1\top} \mathbf{B}^{-1} \mathbf{x}}$. This is a very nice norm for this particular basis, because with respect to this inner product the basis $\mathbf{B}$ is indeed orthonormal, so the basis is as 'good' as possible, as argued above. This shows that for any basis there exist norms for which lattice problems as described above will become relatively easy. See Figure 2 for two different bases with their associated quadratic form norms.

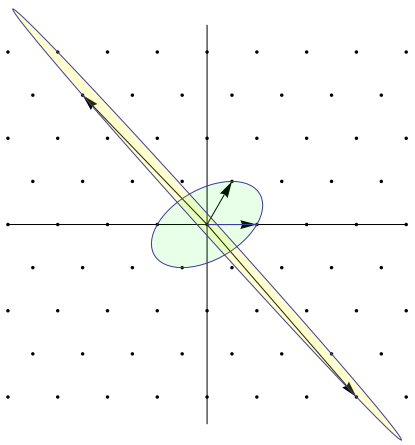

Figure 2: A lattice with unit balls for two different quadratic form norms.

But of course, for all practical purposes, this is not playing fair. In practice one starts with a fixed norm (usually Euclidean), and a lattice is given in terms of a basis that is often quite bad with respect to this given norm. Then, in order to solve lattice problems as described above, one usually needs to first find another basis of the given lattice that is 'good' with respect to the given norm. This will lead us to the idea of 'lattice basis reduction'. Nevertheless it is at least of some historical interest to note that the theory of lattices originated as the theory of positive definite quadratic forms [La], [H].

From now on, for convenience, we will consider only full rank lattices in $\mathbb{Z}^d$, thus of rank d, with $\mathrm{vol}(L) \gg 1$, and only the Euclidean norm.

In Sections 4 and 5 we will see that there exist algorithms that, on input of any basis of a lattice, however bad, compute a 'good' basis $\mathbf{B}$, with a very precise definition of 'good'. The LLL-algorithm provably reaches (roughly) $\prod_i \|\mathbf{b}_i\|_2 / \mathrm{vol}(L) \leq q^{d^2}$ with $q = \sqrt[4]{4/3} + \varepsilon \approx 1.075$, see Section 4.2. Experimentally, for large dimensions and random lattices, lattice bases reduction algorithms (LLL, BKZ) even reach this bound with q as small as 1.013 [N, pp. 52–53]. Heuristically, no algorithm can do better than $d^{d/8}$ [N, p. 53].

2.4 Gram-Schmidt orthogonalization

As argued above, lattice basis reduction methods try to find good bases from bad ones, and a basis is good when it is as orthogonal as possible. Seeking inspiration from linear algebra then leads one to study the main method for orthogonalization, viz. the well known Gram-Schmidt method. One soon discovers that in the lattice setting it miserably fails, as the process encounters independent points that have perfect orthogonality properties but with overwhelming probability will lie outside the lattice. Indeed, almost all lattices do not even admit orthogonal bases, so why even bother? Nevertheless, Gram-Schmidt orthogonalization remains a major ingredient of lattice basis reduction methods, so we do briefly describe it here.

Let $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ be a full rank basis of a lattice L in $\mathbb{Z}^d$. We recursively define a basis $\mathbf{B}^* = \{\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*\}$ by $\mathbf{b}_1^* = \mathbf{b}_1$ and $\mathbf{b}_i^* = \mathbf{b}_i - \mathrm{proj}_i(\mathbf{b}_i)$ for $i \geq 2$, where proj_i is the orthogonal projection onto the real linear span of $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$. These $\mathbf{b}_i^*$ are called the *GSO-vectors* of the basis $\mathbf{B}$. Note that the GSO-vectors depend on the order of the basis vectors, i.e., a different ordering of the basis vectors leads to completely different GSO-vectors $\mathbf{B}^*$. Clearly $\mathbf{B}^*$ is an orthogonal basis, but with overwhelming probability $L(\mathbf{B}^*) \neq L(\mathbf{B})$, and most likely $L(\mathbf{B}^*)$ is not even in $\mathbb{Z}^d$. The formula for the projection is $\mathrm{proj}_i(\mathbf{b}_i) = \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$, where the coefficients $\mu_{i,j}$ are the *Gram-Schmidt coefficients*, defined by $\mu_{i,j} = (\mathbf{b}_i \cdot \mathbf{b}_j^*)/\|\mathbf{b}_j^*\|^2$ for $1 \leq j < i \leq d$. We define the *GS-transformation matrix* as

$$\mathbf{G_B} = \begin{pmatrix} 1 & \mu_{2,1} & \ldots & \ldots & \mu_{d,1} \\ 0 & 1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 1 & \mu_{d,d-1} \\ 0 & \ldots & \ldots & 0 & 1 \end{pmatrix}.$$

Clearly it satisfies $\mathbf{B}^*\mathbf{G_B} = \mathbf{B}$. As $\det(\mathbf{G_B}) = 1$ and $\mathbf{B}^*$ is orthogonal, $\mathrm{vol}(L) = \prod_i \|\mathbf{b}_i^*\|$.

2.5 Minkowski's successive minima and Hermite's constants

Any lattice L of rank at least 1 has, by its discreteness property, a nonzero lattice point that is closest to the origin. Its norm is the shortest distance possible between any two lattice points, and is called the *first successive minimum* $\lambda_1(L)$ of the lattice. In other words, $\lambda_1(L) = \min\{\|\mathbf{x}\| \mid \mathbf{x} \in L, \mathbf{x} \neq \mathbf{0}\}$. Minkowski introduced this concept [Min], and generalized it to other successive minima, as follows.

Definition 2.2. For $i = 1, \ldots, d$ the ith successive minimum of the lattice L of rank d is

$$\lambda_i(L) = \min\{\max\{\|\mathbf{x}_1\|, \ldots, \|\mathbf{x}_i\|\} \mid \mathbf{x}_1, \ldots, \mathbf{x}_i \in L \text{ are linearly independent}\}.$$

Another way to define $\lambda_i(L)$ is as $\min\{r > 0 \mid \dim \mathrm{span}(L \cap rU) \geq i\}$, where U is the unit ball in the Euclidean norm.

One might think that any lattice should have a basis of which the norms are precisely the successive minima, but this turns out to be wrong. It may happen that all independent sets of lattice points of which the norms are precisely the successive minima generate proper sublattices [N, pp. 32–33].

Hermite discovered that $\lambda_1(L)$ can be upper bounded by a constant times $\mathrm{vol}(L)^{1/d}$, where the constant depends only on the rank d, and not on the lattice L. We do not treat Hermite's constants in this paper, see Nguyen's paper [N, pp. 33–35] for a brief account. There is no similar result for higher successive minima on their own, but Hermite's result can be generalized for products of the first k successive minima: $\prod_{i=1}^{k} \lambda_i(L)$ can be upper bounded by a constant times $\mathrm{vol}(L)^{k/d}$, for $k = 1, \ldots, d$.

2.6 Size-reduction

We now turn to an algorithmic viewpoint. The first idea that comes to mind in studying algorithms for lattice basis reduction is to get inspiration from the Euclidean Algorithm, since this algorithm is easily adapted to deal with lattice basis reduction in the rank 2 case. It iterates two steps: size-reduction and swapping. Size-reduction of a basis $\{\mathbf{b}_1, \mathbf{b}_2\}$ with, without loss of generality, $\|\mathbf{b}_1\| \leq \|\mathbf{b}_2\|$, means replacing $\mathbf{b}_2$ by $\mathbf{b}_2 - \lambda \mathbf{b}_1$, for an optimally chosen integer λ such that the new $\mathbf{b}_2$ becomes as short as possible. Iteration stops when size-reduction did not yield any improvement anymore, i.e. when $\|\mathbf{b}_1\| \geq \|\mathbf{b}_2\|$ happens. Note that by the integrality of λ the two bases generate the same lattice at every step in the algorithm. This lattice basis reduction algorithm, essentially the Euclidean algorithm but in the case of 2-dimensional bases known as Lagrange's Algorithm and as Gauss' Algorithm, is further presented and discussed in Section 4.1 as Algorithm 3.

A generalization to higher dimensions is not immediate, because in both the size-reduction and the swapping steps there are many basis vectors to choose from, and one needs a procedure to make this choice. For the size-reduction step this is easy to solve: the $\mu_{i,j}$ in the GS-transformation matrix are approximately the proper λ's. But we have to take into account that 1) we need to subtract only integer multiples whereas $\mu_{i,j}$ probably are not integral, and 2) when $\mathbf{b}_i$ changes, then so will all $\mu_{i,k}$. The following algorithm does this size-reduction in the proper efficient way, treating the columns from left to right, and per column bottom up. In this way the Gram-Schmidt vectors $\mathbf{b}_i^*$ do not change at all.

Algorithm 1 A size-reduction algorithm

Require: a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L
Ensure: the Gram-Schmidt coefficients of the output basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L satisfy $|\mu_{i,j}| \leq \frac{1}{2}$ for each $1 \leq j < i \leq d$

```
for i = 2 to d do
    for j = i − 1 down to 1 do
        b_i ← b_i − ⌊μ_{i,j}⌉ b_j
        for k = 1 to j do
            μ_{i,k} ← μ_{i,k} − ⌊μ_{i,j}⌉ μ_{j,k}
        end for
    end for
end for
```

An obvious way to generalize the swapping step is to order the basis vectors from shortest to longest . This straightforward generalization of the Euclidean algorithm seems not to be the right one, as it is similar to the greedy lattice basis reduction algorithm of Nguyen and Stehlé [NS1], which is known to be far from optimal. In Section 4 we will see what better algorithms exist.

2.7 Babai's algorithms

Finding short nonzero points in a lattice can be done, as discussed above, with a lattice basis reduction algorithm followed by a brute force search. Lattice basis reduction is important here, because as we saw, it greatly improves the efficiency of the search phase. Another natural problem to look at is, given a target vector $\mathbf{t} \in \mathbb{R}^d$, find a lattice vector close to it, preferably the closest one. It can be seen as the inhomogeneous version of the problem of finding short nonzero lattice vectors. For solving this problem it also makes sense to first apply a lattice basis reduction algorithm, in order to get a 'good', almost orthogonal basis of the lattice. To find a lattice vector close to $\mathbf{t}$, Babai [B] introduced two methods described below. Once a lattice point close to $\mathbf{t}$ is found, others (say within a certain ball around $\mathbf{t}$, or the closest one) can be found by brute force search techniques as described above.

The first, quite simple method of Babai is a rounding technique. The idea is to simply compute the coordinates of $\mathbf{t}$ with respect to the reduced lattice basis, and round those to the nearest integer. In a short formula:

when $\mathbf{B}$ is the reduced basis of the lattice, compute $\mathbf{u} = \mathbf{B}\lfloor \mathbf{B}^{-1}\mathbf{t} \rceil$ as a lattice point close to $\mathbf{t}$. Babai proved that $\|\mathbf{u}-\mathbf{t}\| \leq (1+2d(9/2)^{d/2}) \min\{\|\mathbf{z}-\mathbf{t}\| \mid \mathbf{z} \in L(\mathbf{B})\}$, where d is the rank of $L(\mathbf{B})$.

The second method of Babai is the *nearest plane algorithm*. If the reduced basis of the lattice is $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$, then first find $\lambda_d \in \mathbb{Z}$ such that the hyperplane $\lambda_d \mathbf{b}_d + \text{span}(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$ is closest to $\mathbf{t}$. Then replace $\mathbf{t}$ by $\mathbf{t} - \lambda_d \mathbf{b}_d$, and recursively apply the same method to the basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}\}$. The resulting lattice vector is $\sum_i \lambda_i \mathbf{b}_i$. This idea is written out in Algorithm 2. Babai showed that this algorithm achieves $\|\mathbf{u}-\mathbf{t}\| \leq 2(\sqrt{4/3})^d \min\{\|\mathbf{z}-\mathbf{t}\| \mid \mathbf{z} \in L(\mathbf{B})\}$.

Algorithm 2 Babai's nearest plane algorithm

Require: a reduced basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L, and a target vector $\mathbf{t} \in \mathbb{R}^d$
Ensure: the ouput lattice vector $\mathbf{u} \in L$ is 'close' to $\mathbf{t}$

```
s ← t
for i = d down to 1 do
    s ← s − λ b_i, where λ = ⌊(s · b_i^*)/(b_i^* · b_i^*)⌉
end for
u ← t − s
```

2.8 The Hermite factor and high dimensions

To measure the quality of a lattice basis reduction algorithm, one could look only at the length of the first reduced basis vector. Of course this makes sense only if one is only interested in finding one short nonzero lattice vector, and if the first basis vector is (probably) the shortest of the output basis vectors. The latter is often the case with the lattice basis reduction algorithms used in practice.

The ideal measure to look at would be $\|\mathbf{b}_1\|/\lambda_1(L)$, comparing the length of the first basis vector as a candidate short lattice vector to the actual shortest length of a vector in the lattice. This ratio is called the *approximation factor*. However, in practice, given only a (reduced) basis, one does not yet know $\lambda_1(L)$, and computing it can be a very difficult task, i.e., possibly as hard as actually finding a shortest vector.

Therefore, in practice it is much easier to replace $\lambda_1(L)$ by $\text{vol}(L)^{1/d}$ (where d is the rank of L), which is easy to compute from a basis, as $\text{vol}(L)$ simply is its determinant. The resulting ratio $\|\mathbf{b}_1\|/\text{vol}(L)^{1/d}$ is called the *Hermite factor*.

The LLL-algorithm (see Section 4.2) provably reaches a Hermite factor of $(\sqrt{4/3}+\varepsilon)^{(d-1)/2} \approx 1.075^{d-1}$ [LLL]. In practical experiments, for high dimensions, it even appears to reach 1.02^{d-1} [GN]. The algorithm BKZ-20 (see Section 4.4) even reaches $1.012^{(d-1)/2}$ [GN].

At first sight these results seem to be pretty good. For small dimensions this is indeed the case, as the Hermite factors stay close to 1, making the brute force search procedures quite efficient. However, when the dimension d grows, the Hermite factor 1.012^{d-1} grows, first almost linearly like $1+0.012d$, but when d grows the exponential behaviour takes over more and more. For $d = 500$ the factor is already larger than 450, which cannot be called close to 1 any more. Moreover, as this factor describes the radius of the ball in which one has to do the search, and brute force search by its very nature is exponential in the radius, it will become clear that even a Hermite factor very close to 1 (like a hypothetical 1.0001) will for high enough dimensions turn out to become a real obstacle, leading to superexponential search complexity. These heuristics should provide the reader with some crude intuition why lattice problems are thought to be hard problems, so that cryptosystems can be based on them. Section 6 elaborates on these issues.

3 Hard Lattice Problems

This section lists the main *Hard Lattice Problems* on which cryptographic systems have been based. This means that there is a direct relation between the efficiency of solution methods for such a hard problem,

and cryptanalysis of the cryptosystem. If both the hardness of the lattice problem and the relation to attack methods for the cryptosystem are well enough understood, key size recommendations may be established (or, in the worst case, the cryptosystem may be considered completely broken). The focus in this section is on the hard lattice problems themselves, not on cryptosystems and relations between the two. An overview of hard lattice problems and their interconnections is presented. This section is loosely based on [Pol, Sections 1.4, 1.6, 3.1]

In this section, L is a full rank lattice in $\mathbb{Z}^d$.

3.1 Finding short vectors

We recall the *first successive minimum* of the lattice L, which is the length of the shortest nonzero lattice vector (or, stated differently, the minimal distance between two lattice points), i.e., $\lambda_1(L) = \min\{\|\mathbf{x}\| \mid \mathbf{x} \in L, \mathbf{x} \neq \mathbf{0}\}$. The first obvious problem in lattice theory is to find a nonzero lattice vector that reaches this minimum. Note that it is never unique, as $\|(-\mathbf{x})\| = \|\mathbf{x}\|$ for all lattice vectors $\mathbf{x} \in L$ (there may be other lattice points with the same norm).

SVP – The Shortest Vector Problem
Given a basis of L, find $\mathbf{y} \in L$ such that $\|\mathbf{y}\| = \lambda_1(L)$.

Often applications need only sufficiently short lattice points. This gives rise to the following relaxation of SVP.

SVP$_\gamma$ – The Approximate Shortest Vector Problem
Given a basis of L and an approximation factor $\gamma \geq 1$, find $\mathbf{y} \in L$ such that $0 < \|\mathbf{y}\| \leq \gamma\lambda_1(L)$.

Clearly SVP = SVP$_1$. SVP$_\gamma$ is known to be NP-hard for $\gamma = 2^{\log^{1/2-\varepsilon}(d)} \approx \sqrt{d}$ [Kh].

Note that in the above problems, $\lambda_1(L)$ is not known in advance. Computing it exactly is the problem SLP = SLP$_1$, described below. But, as remarked in Section 2.8, $\mathrm{vol}(L)$ is known, and can be used instead. This gives rise to a 'Hermite' variant of (Approximate) SVP.

HSVP$_\gamma$ – The Hermite Shortest Vector Problem
Given a basis of L and an approximation factor $\gamma > 0$, find $\mathbf{y} \in L$ such that $0 < \|\mathbf{y}\| \leq \gamma\mathrm{vol}(L)^{1/d}$.

As noted in Section 2.8, LLL solves HSVP$_\gamma$ (in polynomial time) for $\gamma = (\sqrt{4/3} + \varepsilon)^{(d-1)/2}$, and in practice it achieves $\gamma = 1.02^d$. With better algorithms even $\gamma = 1.01^d$ is within reach (see Section 4.4).

Next we mention a decision variant of SVP.

DSVP – The Decision Shortest Vector Problem
Given a basis of L and a radius $r > 0$, decide whether there exists a $\mathbf{y} \in L$ such that $0 < \|\mathbf{y}\| \leq r$.

Is it possible to find the first successive minimum without actually finding a shortest nonzero lattice vector? This problem is not necessarily equivalent to SVP, thus we have the following hard problem.

SLP$_\gamma$ – The Approximate Shortest Length Problem
Given a basis of L and an approximation factor $\gamma > 1$, find a λ such that $\lambda_1(L) \leq \lambda \leq \gamma\lambda_1(L)$.

In cryptographic applications there is interest in lattices with gaps in the sequence of successive minima. This means that there are smaller rank sublattices with smaller volumes than expected from random lattices, say $L' \subset L$ with rank $d' < d$, such that $\mathrm{vol}(L')^{1/d'}$ is essentially smaller than $\mathrm{vol}(L)^{1/d}$. In other words, there exist lattice vectors in L that are shorter than expected from random lattices. When the gap is between the first two successive minima, the approximate SVP has a separate name.

USVP$_\gamma$ – The Unique Shortest Vector Problem
Given a basis of L and a gap factor $\gamma \geq 1$, find (if it exists) the unique nonzero $\mathbf{y} \in L$ such that any $\mathbf{v} \in L$ with $\|\mathbf{v}\| \leq \gamma \|\mathbf{y}\|$ is an integral multiple of $\mathbf{y}$.

And finally, we mention a problem known as GapSVP.

GapSVP$_\gamma$ – The Gap Shortest Vector Problem
Given a basis of L, a radius $r > 0$ and an approximation factor $\gamma > 1$, return YES if $\lambda_1(L) \leq r$, return NO if $\lambda_1(L) > \gamma r$, and return either YES or NO otherwise.

Such a problem is called a *promise* problem. GapSVP$_\gamma$ is NP-hard for constant γ [Kh].

3.2 Finding close vectors

As seen in Section 2, the second obvious problem in lattice theory is to find, for a given target point $\mathbf{t} \in \mathbb{R}^d$, a lattice point that is closest to $\mathbf{t}$. Note that it may not be unique, but in many cases it will. It makes sense to assume that $\mathbf{t} \notin L$.

Let $d(\mathbf{t}, L)$ denote the distance of $\mathbf{t} \in \mathbb{R}^d$ to the closest lattice point.

CVP – The Closest Vector Problem
Given a basis of L and a target $\mathbf{t} \in \mathbb{R}^d$, find $\mathbf{y} \in L$ such that $\|\mathbf{t} - \mathbf{y}\| = d(\mathbf{t}, L)$.

As with SVP, it is in practice not always necessary to know the exact solution to a CVP-instance; often an approximation suffices. Therefore we have the following relaxation of CVP.

CVP$_\gamma$ – The Approximate Closest Vector Problem
Given a basis of L, a target $\mathbf{t} \in \mathbb{R}^d$ and an approximation factor $\gamma \geq 1$, find $\mathbf{y} \in L$ such that $\|\mathbf{t} - \mathbf{y}\| \leq \gamma d(\mathbf{t}, L)$.

CVP = CVP$_1$ is known to be NP-hard [EB]. CVP$_\gamma$ is known to be NP-hard for any constant γ, and is probably NP-hard for $\gamma = 2^{\log^{1-\varepsilon} d} \approx d$ [ABSS]. Babai's nearest plane algorithm (see Section 2.6) solves CVP$_\gamma$ in polynomial time for $\gamma = 2(\sqrt{4/3})^d$.

There are relations between SVP and CVP. Indeed, SVP can be seen as CVP in suitable sublattices [GMSS]. Given a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$, consider the sublattice generated by the basis where $\mathbf{b}_j$ is replaced by $2\mathbf{b}_j$. Then SVP in the original lattice is essentially the CVP-instance for the target $\mathbf{b}_j$ in the sublattice, for some j. It follows that CVP$_\gamma$ is at least as hard as SVP$_\gamma$.

On the other hand, there is an embedding technique [GGH2] that heuristically converts instances of CVP into instances of SVP. The instances are in different lattices with slightly different ranks but identical volumes. The idea is to expand the basis vectors by a 0-coordinate, and the target vector by a 1-coordinate, and then append the target vector to the basis. Heuristically solving SVP in the expanded lattice solves approximately CVP in the original lattice.

As with SVP there are a number of variants of CVP. First we mention the decision variant.

DCVP – The Decision Closest Vector Problem
Given a basis of L, a target vector $\mathbf{t} \in \mathbb{R}^d$ and a radius $r > 0$, decide whether there exists a $\mathbf{y} \in L$ such that $\|\mathbf{y} - \mathbf{t}\| \leq r$.

There exist cryptosystems, that appear to be based on CVP, but where it is known that the distance between the lattice and the target point is bounded. This leads to the following special case of CVP, which is easier.

BDD$_\alpha$ – Bounded Distance Decoding
Given a basis of L, a distance parameter $\alpha > 0$ and a target vector $\mathbf{t} \in \mathbb{R}^d$ such that $d(\mathbf{t}, L) < \alpha \lambda_1(L)$, find a $\mathbf{y} \in L$ such that $d(\mathbf{y}, \mathbf{t}) = d(L, \mathbf{t})$.

BDD_α becomes harder for increasing α, and it is known to be NP-hard for $\alpha > \frac{1}{2}\sqrt{2}$ [LLM].

3.3 Finding short sets of vectors

As the first successive minimum has a straightforward generalization into a full sequence of successive minima, the SVP has a straightforward generalization as well. We recall their definition:
$\lambda_i(L) = \min\{\max\{\|\mathbf{x}_1\|, \ldots, \|\mathbf{x}_i\|\} \mid \mathbf{x}_1, \ldots, \mathbf{x}_i \in L \text{ are linearly independent}\}$.

SMP – The Successive Minima Problem
Given a basis of L, find a linearly independent set $\{\mathbf{y}_1, \ldots, \mathbf{y}_d\}$ in L such that $\|\mathbf{y}_i\| = \lambda_i(L)$ for $i = 1, \ldots, d$.

For an approximation factor $\gamma > 1$ it is clear how SMP_γ can be defined.

A different problem is to find a shortest basis.

SBP_γ – The Approximate Shortest Basis Problem
Given a basis of L and an approximation factor $\gamma \geq 1$, find a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L with $\max_i \|\mathbf{b}_i\| \leq \gamma \min\{\max_i \|\mathbf{a}_i\| \mid \{\mathbf{a}_1, \ldots, \mathbf{a}_d\} \text{ is a basis of } L\}$.

Again we can define SBP = SBP_1.

A problem closely related to SBP is the following.

SIVP_γ – The Shortest Independent Vector Problem
Given a basis of L and an approximation factor $\gamma > 1$, find a linearly independent set $\{\mathbf{y}_1, \ldots, \mathbf{y}_d\}$ such that $\max_i \|\mathbf{y}_i\| \leq \gamma \lambda_d(L)$.

SIVP_γ is NP-hard for $\gamma = d^{1/\log\log d}$ [BS].

3.4 Modular lattices

To introduce the final few hard problems some more notation is needed. A lattice $L \subset \mathbb{Z}^m$ is called *modular* with modulus q, or *q-ary*, if $q\mathbb{Z}^m \subset L$. Such lattices are only of interest if $q \ll \text{vol}(L)$. Of interest to us are modular lattices of the form $L_{\mathbf{A},q} = \{\mathbf{x} \in \mathbb{Z}^m \mid \mathbf{A}\mathbf{x} \equiv 0 \pmod q\}$, where $\mathbf{A}$ is an $n \times m$ matrix with integer coefficients taken modulo q.

The Small Integer Solutions problem comes from Micciancio and Regev [MR1].

SIS – Small Integer Solutions
Given a modulus q, a matrix $\mathbf{A} \pmod q$ and a $\nu < q$, find $\mathbf{y} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{y} \equiv 0 \pmod q$ and $\|\mathbf{y}\| \leq \nu$.

Although this is not formulated here as a lattice problem, it is clear that it is a lattice problem in the lattice $L_{\mathbf{A},q}$.

Let q be a modulus. For $\mathbf{s} \in \mathbb{Z}_q^n$ and a probability distribution χ on $\mathbb{Z}_q$, let $A_{\mathbf{s},\chi}$ be the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ with sampling as follows: take $\mathbf{a} \in \mathbb{Z}_q^n$ uniform, take $e \in \mathbb{Z}_q$ according to χ, then return $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \pmod q$.

The Learning with Errors problem was posed by Regev [R1].

LWE – Learning With Errors
Given n, q, χ and any number of independent samples from $A_{\mathbf{s},\chi}$, find $\mathbf{s}$.

For χ the discrete Gaussian distribution is a good choice.

For the sampled $(\mathbf{a}_i, b_i)$ let $\mathbf{A}$ be the matrix of the $\mathbf{a}_i$, and let $\mathbf{b}$ be the vector of the b_i. Then $\{\mathbf{x} \mid \mathbf{x} \equiv \mathbf{A}^\top \mathbf{y}$

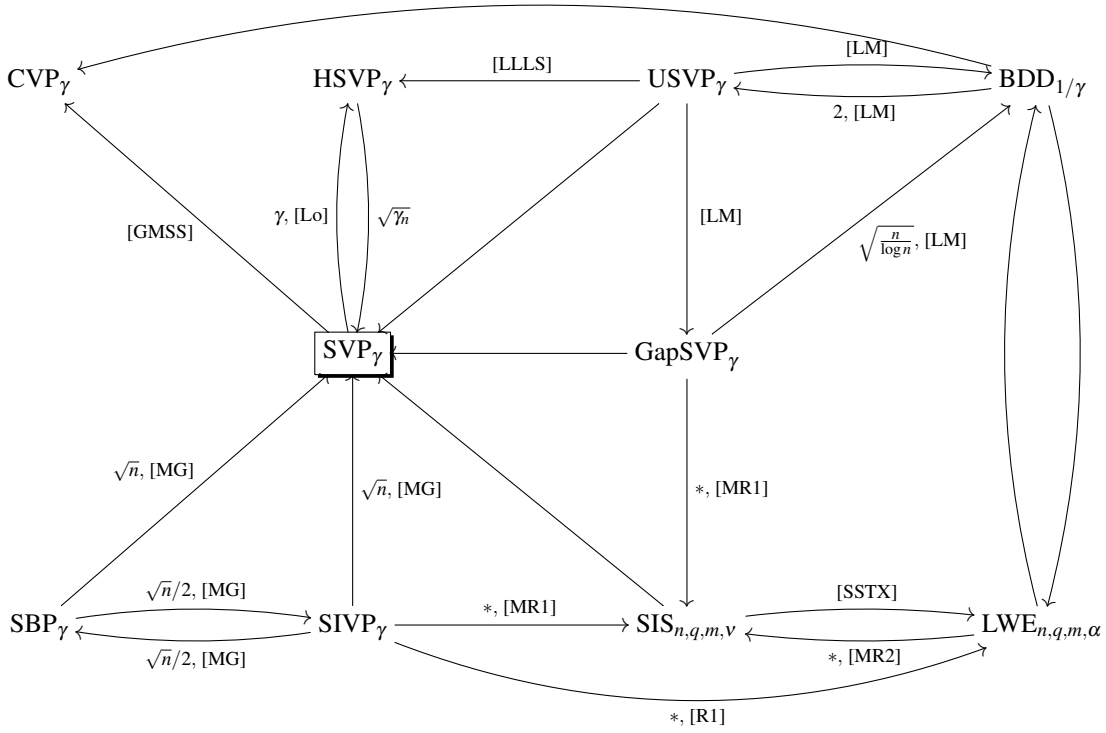

Figure 3: Relations among lattice problems

$(\mathrm{mod}\ q)$ for some $\mathbf{y}\}$ is a lattice in which $\mathbf{A}^\top\mathbf{s}$ is close to $\mathbf{b}$, so this is a CVP-variant, in fact, a BDD-variant.

DLWE – Decision Learning With Errors
Given n, q, χ and any number of independent samples from $A_{\mathbf{s},\chi}$, return YES if the samples come from $A_{\mathbf{s},\chi}$, and NO if they come from the normal distribution.

3.5 Reductions between hard problems

To end this section, we indicate relations between the main hard lattice problems presented above. The difficulty of hard problems can sometimes be compared by *reductions* between those problems. We say that Problem A reduces to Problem B if any method that solves all instances of Problem B can be used to solve the instances of Problem A. If this is the case, then Problem B cannot be easier than Problem A. Said differently, Problem B is at least as hard as Problem A.

Micciancio and Goldwasser [MG] presented a graphical representation of the relations between the classical Hard Lattice Problems. This inspired us to make a similar picture for what we see as the main Hard Lattice Problems of today, see Figure 3. In this picture, an arrow from Problem A to Problem B means that Problem A can be reduced to Problem B in polynomial time.

A subscript γ is the approximation factor for SVP, CVP, HSVP, SBP and SIVP, the gap for GapSVP and USVP, and for BDD the subscript $1/\gamma$ is the distance bound.

An arrow with a label α means that the reduction loses a factor α in the subscript. The reductions marked with an asterisk have some more technical conditions attached to them. See Van de Pol's thesis [Pol, Section 3.1.6] for more details on these reductions, and for references to the relevant literature.

An interesting observation we can make from this picture is that the Approximate Shortest Vector Problem SVP$_\gamma$ has many incoming arrows. Apart from CVP$_\gamma$, all problems can (indirectly) be reduced to it, and

even the reduction of CVP_γ to SVP_γ is arguable by the embedding technique. This can be interpreted as SVP_γ being the central hard lattice problem. Therefore in Section 4, algorithms for solving SVP_γ will be the main topic of study.

4 Solving the Approximate Shortest Vector Problem

As we saw in the previous section, the Approximate Shortest Vector Problem is one of the most important hard lattice problems, as many other hard lattice problems can be reduced to this problem. Moreover, as we will see in Section 6, many cryptosystems rely on the hardness of finding (reasonably) short vectors in lattices. Being able to find vectors with sufficiently small approximation factors in these lattices would mean being able to break these cryptosystems. So to estimate the security of these cryptosystems, it is essential to understand algorithms for solving approximate SVP, and to be able to quantify their performance for the lattices arising from those cryptosystems.

In this section we will look at several algorithms for solving approximate SVP. The algorithms described in this section are all *lattice basis reduction algorithms*; instead of outputting a single short vector, these algorithms produce an entire basis of reasonably short vectors. However, to evaluate the quality of the algorithms, we will focus on the length of the shortest vector of the output basis (typically $\mathbf{b}_1$), and the associated approximation factor $\|\mathbf{b}_1\|/\lambda_1(L)$ and Hermite factor $\|\mathbf{b}_1\|/\text{vol}(L)^{1/d}$.

In Section 4.1, we first look at Gauss' algorithm for finding an optimal basis in 2 dimensions. By considering a relaxation and blockwise application of this algorithm, in Section 4.2 we naturally end up with the celebrated LLL algorithm for finding reasonably short vectors in higher dimensions in polynomial time. In Section 4.3, we will then look at KZ-reduction and an algorithm to find KZ-reduced bases in k dimensions, and consider a similar relaxation and blockwise application in Section 4.4, to end up with Schnorr's hierarchy of basis reduction algorithms, and Schnorr and Euchner's famous BKZ algorithm. From the BKZ algorithm, it will also become clear why it is important to analyze algorithms for solving the exact shortest vector problem, which is done in the next section. Finally, in Section 4.5 we give a brief overview of the algorithms discussed in this section, and how they relate to each other.

4.1 Gauss' algorithm

We will start with an easy problem: Finding a basis $\{\mathbf{b}_1, \mathbf{b}_2\}$ for a 2-dimensional lattice L such that $\|\mathbf{b}_1\| = \lambda_1(L)$ and $\|\mathbf{b}_2\| = \lambda_2(L)$. We can solve this problem with *Gauss' algorithm* [Ga], given in Algorithm 3, which is sometimes also attributed to Lagrange [La]. In this algorithm we assume that at any point, the Gram-Schmidt coefficient $\mu_{2,1} = (\mathbf{b}_2 \cdot \mathbf{b}_1)/\|\mathbf{b}_1\|^2$ is known and up to date with respect to the current vectors $\mathbf{b}_1$ and $\mathbf{b}_2$. For simplicity, and to focus on the high-level description rather than the low-level details, in the algorithms discussed in this section we will omit details on updating the GSO-vectors $\mathbf{b}_i^*$ and the coefficients $\mu_{i,j}$.

Algorithm 3 Gauss' basis reduction algorithm

Require: a basis $\{\mathbf{b}_1, \mathbf{b}_2\}$ of L
Ensure: the output basis $\{\mathbf{b}_1, \mathbf{b}_2\}$ of L satisfies $|\mu_{2,1}| \leq \frac{1}{2}$ and $\|\mathbf{b}_1\| \leq \|\mathbf{b}_2\|$
1: $\mathbf{b}_2 \leftarrow \mathbf{b}_2 - \lfloor \mu_{2,1} \rceil \mathbf{b}_1$
2: **while** $\|\mathbf{b}_1\| \geq \|\mathbf{b}_2\|$ **do**
3: swap$(\mathbf{b}_1, \mathbf{b}_2)$
4: $\mathbf{b}_2 \leftarrow \mathbf{b}_2 - \lfloor \mu_{2,1} \rceil \mathbf{b}_1$
5: **end while**

This algorithm is closely related to Euclid's greatest common divisor algorithm [E], as was already mentioned in Section 2.6. At each iteration, we subtract the shortest of the two vectors ($\mathbf{b}_1$) an integal number of times ($\lfloor \mu_{2,1} \rceil = \lfloor \frac{(\mathbf{b}_1 \cdot \mathbf{b}_2)}{\|\mathbf{b}_1\|^2} \rceil$) from the longest of the two ($\mathbf{b}_2$) to obtain a shorter vector $\mathbf{b}_2 \leftarrow \mathbf{b}_2 - \lfloor \mu_{2,1} \rceil \mathbf{b}_1$,

and we swap $\mathbf{b}_1$ and $\mathbf{b}_2$. In Euclid's greatest common divisor algorithm, for inputs a and b with $a < b$, we also subtract the smallest of the two numbers (a) an integral number of times ($\lfloor \mu \rfloor = \lfloor \frac{a \cdot b}{a^2} \rfloor$) from the largest of the two (b) to obtain a smaller number $b \leftarrow b - \lfloor \mu \rfloor a$. But while in Euclid's algorithm we usually continue until $b = 0$, Gauss' algorithm stops 'halfway', as soon as $\|\mathbf{b}_1\| \geq \|\mathbf{b}_2\|$. This variant of the Euclidean algorithm is also known in the literature as the half-GCD algorithm.

When Gauss' algorithm terminates, we know that $|\mu_{2,1}| \leq \frac{1}{2}$, and the resulting basis $\{\mathbf{b}_1, \mathbf{b}_2\}$ then satisfies the properties described below.

Theorem 4.1. *Given a basis $\{\mathbf{b}_1, \mathbf{b}_2\}$ of a lattice L as input, Gauss' algorithm terminates in* $\text{poly}(\log \|\mathbf{B}\|)$ *time (i.e., in time polynomial in the size of the input basis $\mathbf{B}$) and outputs a basis $\{\mathbf{b}_1, \mathbf{b}_2\}$ of L satisfying:*

$$\frac{\|\mathbf{b}_1\|}{\lambda_1(L)} = 1, \quad \frac{\|\mathbf{b}_1\|}{\text{vol}(L)^{1/2}} \leq \sqrt{\gamma_2} = \sqrt[4]{\frac{4}{3}} \approx 1.0746, \quad \frac{\|\mathbf{b}_1^*\|}{\|\mathbf{b}_2^*\|} \leq \sqrt{\frac{4}{3}}. \tag{4.1}$$

Let us briefly explain where the Hermite factor $\gamma_2 = \sqrt{4/3}$ comes from. We know that for the Gram-Schmidt orthogonalization of the output basis $\{\mathbf{b}_1, \mathbf{b}_2\}$, we have $\mathbf{b}_1^* = \mathbf{b}_1$ and $|\mu_{2,1}| \leq \frac{1}{2}$, and we know that the output basis satisfies $\|\mathbf{b}_2\| \geq \|\mathbf{b}_1\|$. So we get a lower bound on the ratio between the norms of $\mathbf{b}_2^*$ and $\mathbf{b}_1^*$ as

$$\frac{\|\mathbf{b}_2^*\|^2}{\|\mathbf{b}_1^*\|^2} = \frac{\|\mathbf{b}_2 - \mu_{2,1}\mathbf{b}_1\|^2}{\|\mathbf{b}_1\|^2} \geq \frac{\|\mathbf{b}_2\|^2 - \mu_{2,1}^2\|\mathbf{b}_1\|^2}{\|\mathbf{b}_1\|^2} \geq 1 - \mu_{2,1}^2 \geq \frac{3}{4}.$$

The result then follows from the fact that $\text{vol}(L) = \|\mathbf{b}_1^*\| \cdot \|\mathbf{b}_2^*\|$.

Note that the upper bounds in (4.1) are sharp for lattices of rank 2 in general, since the *hexagonal lattice*, spanned by the vectors $\mathbf{b}_1 = (1,0)$ and $\mathbf{b}_2 = (\frac{1}{2}, \frac{1}{2}\sqrt{3})$, attains these bound. This lattice is shown in Figures 1 and 2.

4.2 The LLL algorithm

By applying Gauss' algorithm to a basis $\{\mathbf{b}_1, \mathbf{b}_2\}$, we are basically ensuring that the ratio between the two GSO-vectors, $\|\mathbf{b}_2^*\|^2/\|\mathbf{b}_1^*\|^2 \geq 1 - \mu_{2,1}^2 \geq \frac{3}{4}$, cannot be too small. Since the volume of the lattice is the product of the norms of the GSO-vectors, the volume is then bounded from below by a constant times $\|\mathbf{b}_1^*\|^d$. This is also the main idea behind (the proof of) the *Lenstra-Lenstra-Lovász (LLL) algorithm* [LLL], which is essentially a blockwise, relaxed application of Gauss' algorithm to higher-dimensional lattices. Given a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of a d-dimensional lattice L, we want to make sure that for each i from 2 to d, the ratio between the lengths of the consecutive Gram-Schmidt vectors $\mathbf{b}_i^*$ and $\mathbf{b}_{i-1}^*$ is sufficiently large. The most natural choice for a lower bound on these values would be $1 - \mu_{i,i-1}^2 \geq \frac{3}{4}$, but with this choice, no one has been able to find an algorithm with a provable polynomial runtime. To ensure a polynomial time complexity, Lovász therefore introduced a slight relaxation of this condition. For $\delta \in (\frac{1}{4}, 1)$, and for each i between 2 and d, *Lovász' condition* is defined as

$$\frac{\|\mathbf{b}_i^*\|^2}{\|\mathbf{b}_{i-1}^*\|^2} \geq \delta - \mu_{i,i-1}^2. \tag{4.2}$$

The LLL algorithm can be summarized as applying Gauss' algorithm iteratively to each pair of vectors $\mathbf{b}_i, \mathbf{b}_{i-1}$, for i from 2 to d, ensuring that each pair of vectors $\mathbf{b}_i, \mathbf{b}_{i-1}$ satisfies the Lovász condition. The algorithm also makes sure that the basis is size-reduced at all times. When we swap two vectors $\mathbf{b}_i$ and $\mathbf{b}_{i-1}$, we may ruin the relation between $\mathbf{b}_{i-1}^*$ and $\mathbf{b}_{i-2}^*$, so each time we swap two vectors, we decrease i by 1 to see if we need to fix anything for previous values of i. This leads to the LLL algorithm given in Algorithm 4. Note that taking $d = 2$ and $\delta = 1$ leads to Gauss' 2-dimensional reduction algorithm, so the LLL algorithm could be seen as a generalization of Gauss' algorithm.

The description of the LLL algorithm, given in Algorithm 4, is a simplified description of any real implementation of the algorithm, where we left out details on updating the Gram-Schmidt vectors and coefficients. We simply assume that the Gram-Schmidt orthogonalization is always known and up to date with

Algorithm 4 The Lenstra-Lenstra-Lovász (LLL) basis reduction algorithm

Require: a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L, and a constant $\delta \in (\frac{1}{4}, 1)$
Ensure: the output basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L satisfies (4.2) for each i from 2 to d
1: $i \leftarrow 2$
2: **while** $i \leq d$ **do**
3: $\quad \mathbf{b}_i \leftarrow \mathbf{b}_i - \sum_{j=1}^{i-1} \lfloor \mu_{i,j} \rceil \mathbf{b}_j$
4: $\quad$ **if** $\|\mathbf{b}_i^*\|^2 \geq (\delta - \mu_{i,i-1}^2)\|\mathbf{b}_{i-1}^*\|^2$ **then**
5: $\quad\quad i \leftarrow i+1$
6: $\quad$ **else**
7: $\quad\quad$ swap$(\mathbf{b}_i, \mathbf{b}_{i-1})$
8: $\quad\quad i \leftarrow \max\{2, i-1\}$
9: $\quad$ **end if**
10: **end while**

respect to the current basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$, even though in lines 3 (when a basis vector is changed) and 7 (when two basis vectors are swapped), this Gram-Schmidt orthogonalization is changed. In practice, we have to update the values of $\mu_{i,j}$ and $\mathbf{b}_i^*$ accordingly after each swap and each reduction. Fortunately, these updates can be done in polynomial time.

One can prove that the LLL algorithm runs in polynomial time, and achieves certain exponential approximation and Hermite factors. Theorem 4.2 formally describes these results.

Theorem 4.2. *Given a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of a lattice L and a constant $\delta \in (\frac{1}{4}, 1)$ as input, the LLL algorithm terminates in $\mathrm{poly}(d, (1-\delta)^{-1}, \log \|\mathbf{B}\|)$ time and outputs a reduced basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L satisfying:*

$$\frac{\|\mathbf{b}_1\|}{\lambda_1(L)} \leq \left(\frac{1}{\delta - \frac{1}{4}}\right)^{(d-1)/2}, \quad \frac{\|\mathbf{b}_1\|}{\mathrm{vol}(L)^{1/d}} \leq \left(\frac{1}{\delta - \frac{1}{4}}\right)^{(d-1)/4}.$$

In particular, for $\delta = 1 - \varepsilon \approx 1$ for small values of ε, the LLL algorithm terminates in $\mathrm{poly}(d, 1/\varepsilon)$ time and outputs a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ satisfying:

$$\frac{\|\mathbf{b}_1\|}{\lambda_1(L)} \leq \left(\frac{4}{3} + O(\varepsilon)\right)^{(d-1)/2} \approx 1.1547^{d-1}, \quad \frac{\|\mathbf{b}_1\|}{\mathrm{vol}(L)^{1/d}} \leq \left(\frac{4}{3} + O(\varepsilon)\right)^{(d-1)/4} \approx 1.0746^{d-1}.$$

While it may be clear that the Hermite factor comes from repeatedly applying the Lovász condition and using $\mathrm{vol}(L) = \prod_{i=1}^d \|\mathbf{b}_i^*\|$, at first sight it is not easy to see why this algorithm runs in polynomial time. We give a sketch of the proof here. First, let us assume that the basis vectors are all integral, and consider the quantity $N = \prod_{i=1}^d \|\mathbf{b}_i^*\|^{2(d-i+1)}$. Since N can equivalently be written as $N = \prod_{j=1}^d \left(\prod_{i=1}^j \|\mathbf{b}_i^*\|^2\right) = \prod_{j=1}^d \mathrm{vol}(\{\mathbf{b}_1, \ldots, \mathbf{b}_j\})^2$, it follows that $N \in \mathbb{N}$. Now if we investigate what happens when we swap two vectors $\mathbf{b}_i$ and $\mathbf{b}_{i-1}$ in line 7, we notice that this quantity N decreases by a factor of at least δ. It follows that the number of swaps is at most logarithmic in N. Finally, since $N \leq \max_i \|\mathbf{b}_i\|^{2d} = \|\mathbf{B}\|^{2d}$ is at most exponential in d, the number of iterations is at most polynomial in d.

4.2.1 Improvements for the LLL algorithm

Since the publication of the LLL algorithm in 1982, many variants have been proposed, which in one way or another improve the performance. One well-known variant of the LLL algorithm is using *deep insertions*, proposed by Schnorr and Euchner in 1994 [SE]. Instead of swapping the two neighboring basis vectors $\mathbf{b}_i$ and $\mathbf{b}_{i-1}$, this algorithm inserts $\mathbf{b}_i$ somewhere in the basis, at some position $j < i$. This can be seen as a further generalization of the LLL algorithm, since taking $j = i-1$ for each swap leads to the

original LLL algorithm. To choose where to insert the vector $\mathbf{b}_i$, we look for the smallest index j such that inserting the vector at that position, we gain a factor of at least δ, as in the proof of the LLL-algorithm. So we look for the smallest index j such that:

$$\delta \|\mathbf{b}_j^*\|^2 > \|\mathbf{b}_i\|^2.$$

When the algorithm terminates, the notion of reduction achieved is slightly stronger than in the original LLL algorithm, namely:

$$\forall j < i \leq d: \quad \delta \|\mathbf{b}_j^*\|^2 \leq \left\| \mathbf{b}_i^* + \sum_{k=j}^{i-1} \mu_{i,k} \mathbf{b}_k^* \right\|^2.$$

In practice, this slight modification leads to shorter vectors but a somewhat longer runtime. In theory, proving a better performance and proving a polynomial runtime (when the gap between j and i is allowed to be arbitrarily large) seems hard.

Besides theoretical improvements of the LLL algorithm, practical aspects of LLL have also been studied. For example, how does floating-point arithmetic influence the behaviour of the LLL algorithm? Working with floating-point arithmetic is much more efficient than working exactly with rationals, so this is very relevant when actually trying to implement the LLL algorithm efficiently, e.g., as is done in the NTL C++ library [Sho]. When working with floating-point numbers, cancellations of large numbers and loss of precision can cause strange and unexpected results, and several people have investigated how to deal with these problems. For further details, see papers by, e.g., Stehlé [St], Schnorr and Euchner [SE] or Nguyen and Stehlé [NS2, NS4].

Finally, let us emphasize the fact that LLL is still not very well understood. It seems to work much better in practice than theory suggests, and no one really seems to understand why. Several papers have investigated the practical performance of LLL on 'random bases' to find explanations. For instance, Gama and Nguyen [GN] conducted extensive experiments with LLL and LLL with deep insertions (and BKZ, see Section 4.4), and noticed that the Hermite factor converges to approximately 1.02^d for large dimensions d. Nguyen and Stehlé [NS3] studied the configuration of local bases $\{\mathbf{b}_i^*, \mu^{i+1,i}\mathbf{b}_i^* + \mathbf{b}_{i+1}^*\}$ output by the LLL algorithm, and obtained interesting but 'puzzling' results in, e.g., [NS3, Figure 4]. Vallée and Vera [VV] investigated whether the constant 1.02 can be explained by some mathematical formula, but it seems that this problem is still too hard for us to solve. So overall, it is safe to say that several open problems still remain in this area.

4.3 KZ-reduction

In the previous subsections we focused on 2-dimensional reduction: Gauss' algorithm, for finding optimal 2-dimensional bases, and the blockwise LLL algorithm, for finding bases in high dimensions that are locally almost Gauss-reduced. In the next two subsections we will follow the same path, but for the more general k-dimensional setting, for $k \geq 2$. First we consider what optimal means, and we give an algorithm for finding these optimal bases in k dimensions. Then we show how to use this as a subroutine in a blockwise algorithm, to obtain lattice bases in high dimensions with smaller exponential approximation factors than the LLL algorithm in a reasonable amount of time. This provides us with a hierarchy of lattice basis reduction algorithms, with a clear tradeoff between the time complexity and the quality of the output basis.

For finding 'optimal' k-dimensional bases, we first have to decide what kind of optimality we want. To achieve a good notion of reduction, for now we will simply assume that we have access to an oracle $\mathcal{O}$ that solves the shortest vector problem in any lattice of rank at most k; actual algorithms that can act as SVP oracles will be discussed in the next section. Now, an obvious choice of optimality would be to demand that an optimal k-dimensional reduced basis should satisfy $\|\mathbf{b}_i\| = \lambda_i(L)$ for each i from 1 to k. But achieving this seems hard, even with access to SVP oracles. With SVP oracles, we can only find a shortest vector in any lattice, and so a notion of reduction in terms of the first minimum would make more sense. Below

we describe a notion of reduction, together with an algorithm for achieving it, that will give us bases with really small approximation and Hermite factors.

First, with access to an SVP oracle, we can easily let the shortest basis vector $\mathbf{b}_1$ of the output basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ satisfy $\|\mathbf{b}_1\| = \lambda_1(L)$, by choosing the first output basis vector as a shortest vector of L. We then decompose the vectors $\mathbf{v} \in L$ according to $\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2$, with $\mathbf{v}_2 = \alpha_1 \mathbf{b}_1$ a linear combination of $\mathbf{b}_1$, and $\mathbf{v}_2 \in \langle \mathbf{b}_1 \rangle^\perp$ orthogonal to $\mathbf{b}_1$. The set of these vectors $\mathbf{v}_2$ is also denoted by $\pi_2(L)$, the *projected lattice* of L, projected over the complement of the linear span $\langle \mathbf{b}_1 \rangle$. Vectors in $\pi_2(L)$ are generally not in the lattice, but we can always *lift* any vector in $\pi_2(L)$ to a lattice vector, by adding a suitable amount of $\mathbf{b}_1$ to it. As with the size-reduction technique, this suitable amount can always be chosen between $-\frac{1}{2}$ and $+\frac{1}{2}$. Now, a short vector in $\pi_2(L)$ does not necessarily correspond to a short lattice vector, but we do know that for size-reduced bases, the following inequality holds:

$$\|\mathbf{b}_i\|^2 = \left\| \mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^* \right\|^2 \leq \|\mathbf{b}_i^*\|^2 + \sum_{j=1}^{i-1} |\mu_{i,j}|^2 \|\mathbf{b}_j^*\|^2 \leq \|\mathbf{b}_i^*\|^2 + \frac{1}{4} \sum_{j=1}^{i-1} \|\mathbf{b}_j^*\|^2. \tag{4.3}$$

Instead of finding a vector achieving the second successive minimum of L, let us now use the SVP oracle on the lattice $\pi_2(L)$ to find a shortest vector in this projected lattice, and let us treat this shortest vector as the projection $\mathbf{b}_2^*$ of some lattice vector $\mathbf{b}_2$. Since the lifting makes sure that the basis $\{\mathbf{b}_1, \mathbf{b}_2\}$ is size-reduced and $\mathbf{b}_1 = \mathbf{b}_1^*$ is a shortest vector of L, we can use (4.3) to get an upper bound on the length of the lifted vector $\mathbf{b}_2$ as follows:

$$\|\mathbf{b}_2\|^2 \leq \|\mathbf{b}_2^*\|^2 + \frac{1}{4}\|\mathbf{b}_1^*\|^2 = \lambda_1^2(\pi_2(L)) + \frac{1}{4}\lambda_1^2(L).$$

Since the shortest vector in the projected lattice $\lambda_1(\pi_2(L))$ is never longer than the vector achieving the second minimum of the lattice $\lambda_2(L)$, we have $\lambda_1(L), \lambda_1(\pi_2(L)) \leq \lambda_2(L)$. So dividing both sides by $\lambda_2^2(L)$ and using these lower bounds on $\lambda_2(L)$, we get:

$$\frac{\|\mathbf{b}_2\|^2}{\lambda_2^2(L)} \leq \frac{4}{3}.$$

After finding $\mathbf{b}_2$ with $\|\mathbf{b}_2\| = \lambda_1(\pi_2(L))$, we repeat the above process with $\pi_3(L)$, consisting of lattice vectors projected on $\langle \mathbf{b}_1, \mathbf{b}_2 \rangle^\perp$. We first use an SVP oracle to find a vector $\mathbf{b}_3^* \in \pi_3(L)$ with $\|\mathbf{b}_3^*\| = \lambda_1(\pi_3(L))$, and we then lift it to a lattice vector $\mathbf{b}_3 \in L$. Applying (4.3) and $\lambda_1(L), \lambda_1(\pi_2(L)), \lambda_1(\pi_3(L)) \leq \lambda_3(L)$ then gives us $\|\mathbf{b}_3\|^2 \leq \frac{6}{4}\lambda_3^2(L)$. Repeating this procedure for i from 4 to k, we thus obtain a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ with, for each i, $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(L))$, and:

$$\frac{\|\mathbf{b}_i\|^2}{\lambda_i^2(L)} \leq \frac{i+3}{4}. \tag{4.4}$$

This notion of reduction, where the Gram-Schmidt vectors of a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ satisfy $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(L))$ for each i, is also called *Korkine-Zolotarev (KZ) reduction* [KZ], or *Hermite-Korkine-Zolotarev (HKZ) reduction*, and the bases are called KZ-reduced. The procedure described above, to obtain KZ-reduced bases, is summarized in Algorithm 5. Note that while the LLL algorithm was named after the inventors of the *algorithm*, here the *notion of reduction* is named after Korkine and Zolotarev. Algorithm 5 is just an algorithm to achieve this notion of reduction.

The following theorem summarizes the quality of the first basis vector, and the ratio between the lengths of the first and last Gram-Schmidt vectors of any KZ-reduced basis. This will be useful for Section 4.4.

Theorem 4.3. *Given a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ of a lattice L and an SVP oracle $\mathcal{O}$ for up to k dimensions, the Korkine-Zolotarev reduction algorithm terminates after at most k calls to the SVP oracle $\mathcal{O}$ and outputs a reduced basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ of L satisfying:*

$$\frac{\|\mathbf{b}_1\|}{\lambda_1(L)} = 1, \quad \frac{\|\mathbf{b}_1\|}{\mathrm{vol}(L)^{1/k}} \leq \sqrt{\gamma_k} = O(\sqrt{k}), \quad \frac{\|\mathbf{b}_1^*\|}{\|\mathbf{b}_k^*\|} \leq k^{(1+\ln k)/2}.$$

where H_k is the set of all k-dimensional KZ-reduced bases.

Algorithm 5 A Korkine-Zolotarev (KZ) basis reduction algorithm

Require: a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ of L, and an SVP-oracle $\mathcal{O}$ for up to k dimensions
Ensure: the output basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ of L satisfies $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(L))$ for each $i \in \{1, \ldots, k\}$
1: **for** $i = 1$ to k **do**
2: call the SVP oracle $\mathcal{O}$ to find a vector $\mathbf{b}_i^* \in \pi_i(L)$ of length $\lambda_1(\pi_i(L))$
3: lift $\mathbf{b}_i^*$ into a lattice vector $\mathbf{b}_i$ such that $\{\mathbf{b}_1, \ldots, \mathbf{b}_i\}$ is size-reduced
4: replace the basis vectors $\{\mathbf{b}_{i+1}, \ldots, \mathbf{b}_k\}$ by lattice vectors $\{\mathbf{b}_{i+1}, \ldots, \mathbf{b}_k\}$ such that $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ is a basis for L
5: **end for**

Note that finding KZ-reduced bases is at least as hard as finding a shortest vector in k dimensions, since the shortest basis vector is a shortest vector of the lattice. So in high dimensions this algorithm and this notion of reduction are impractical. This algorithm only terminates in a reasonable amount of time when k is sufficiently small. If we want to find nice bases for arbitrary d-dimensional lattices, for high d, we need different methods.

4.4 The BKZ algorithm

As in Section 4.2, it turns out that we can use the KZ-reduction algorithm as a subroutine for finding nice d-dimensional bases. If we can make sure that every block of k consecutive basis vectors is KZ-reduced, then we can prove strong bounds on the length of the first basis vector. More precisely, if for each i from 1 to $d-k+1$, the lattice $\pi_i(\{\mathbf{b}_i, \ldots, \mathbf{b}_{i+k-1}\})$ (spanned by the vectors $\mathbf{b}_i, \ldots, \mathbf{b}_{i+k-1}$ projected on $\langle \mathbf{b}_1, \ldots, \mathbf{b}_{i-1} \rangle^{\perp}$) is KZ-reduced, then the first basis vector $\mathbf{b}_1$ satisfies: [Sno1, Theorem 2.3]

$$\frac{\|\mathbf{b}_1\|}{\lambda_1(L)} \leq \alpha_k^{(d-1)/(2k-2)}. \tag{4.5}$$

Proving this is done by comparing the lengths of the pairs of vectors $\mathbf{b}_{i(k-1)}$ and $\mathbf{b}_{(i+1)(k-1)-1}$, for i ranging from 1 to $\frac{d-1}{k-1}$. For each pair, we use the fact that the block containing these two vectors as first and last vectors is KZ-reduced, to show that their ratio is bounded from above by α_k. The product of these ratios then telescopes to the left hand side of (4.5), while the $\frac{d-1}{k-1}$ factors α_k lead to the right hand side of (4.5), proving the result.

To get an absolute upper bound on the quality of the first basis vector, we also need a bound on α_k. Schnorr provided one in his 1987 paper [Sno1, Corollary 2.5], by showing that $\alpha_k \leq k^{1+\ln k}$ for all $k \geq 2$. This means that if we can achieve the notion of reduction where each local k-block is KZ-reduced and the whole basis is LLL-reduced, then the first basis vector will satisfy

$$\frac{\|\mathbf{b}_1\|}{\lambda_1(L)} \leq \left(k^{\frac{1+\ln k}{2k-2}}\right)^{d-1}.$$

Since $k^{\frac{1+\ln k}{2k-2}} \to 0$ for large k, one could also say that

$$\frac{\|\mathbf{b}_1\|}{\lambda_1(L)} \leq (1+\varepsilon_k)^{d-1},$$

where ε_k is a constant that only depends on k and which converges to 0 as k increases. This means that with this notion of reduction, one can achieve arbitrarily small approximation factors, and even find *shortest* vectors for sufficiently large k. Of course, for $k = d$ this is trivial, as then $\|\mathbf{b}_1\| = \lambda_1(L)$.

To get a basis that is locally KZ-reduced, Schnorr and Euchner [SE] proposed the algorithm presented in Algorithm 6, known as the *Block Korkine-Zolotarev (BKZ) algorithm*. This is similar to the LLL algorithm: we simply iterate KZ-reduction on local blocks, until each block is KZ-reduced. By KZ-reducing a block, we may have to go back up to $k-1$ positions, as preceding blocks may no longer be KZ-reduced. The algorithm then repeats these reductions until all k-blocks are KZ-reduced.

Algorithm 6 Schnorr and Euchner's Block Korkine-Zolotarev (BKZ) basis reduction algorithm

Require: a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L, a blocksize k, a constant $\delta \in (\frac{1}{4}, 1)$, and an SVP-oracle $\mathcal{O}$ for up to k dimensions

Ensure: the output basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L is LLL-reduced with factor δ and satisfies $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(\mathbf{b}_i, \ldots, \mathbf{b}_{i+k-1}))$ for each i from 1 to $d-k+1$

1: **repeat**
2: **for** $i = 1$ to $d-k+1$ **do**
3: KZ-reduce the basis $\pi_i(\mathbf{b}_i, \ldots, \mathbf{b}_{i+k-1})$
4: size-reduce the basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$
5: **end for**
6: **until** no changes occur

For LLL, we could prove a polynomial runtime, because we could find an invariant N which always decreased by a factor δ whenever we 'did' something. For the BKZ algorithm, we cannot prove a similar upper bound on the time complexity. The algorithm behaves well in practice, but theoretically it is not known whether (for fixed $k > 2$) the algorithm always terminates in polynomial time. But when the algorithm terminates, we do know that the first basis vector is short.

Theorem 4.4. *Given a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of a lattice L and an SVP oracle $\mathcal{O}$ for up to k dimensions, the Block Korkine-Zolotarev reduction algorithm outputs a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L satisfying:*

$$\frac{\|\mathbf{b}_1\|}{\lambda_1(L)} \leq \left(k^{\frac{1+\ln k}{2k-2}}\right)^{d-1}, \quad \frac{\|\mathbf{b}_1\|}{\mathrm{vol}(L)^{1/d}} \leq \sqrt{\gamma_k} \cdot \left(k^{\frac{1+\ln k}{2k-2}}\right)^{d-1}.$$

4.4.1 Improvements for the BKZ algorithm

Some improvements were suggested for BKZ over the years, most of which involve improving the SVP subroutine. For instance, so far the best results seem to have been obtained by using what Chen and Nguyen called BKZ 2.0 [CN]. They used improved enumeration techniques to speed up the algorithm, and to be able to run BKZ with higher blocksizes than was deemed possible before. For more details on this, see Sections 5 and 6.5.

One notable other suggested improvement, which does not involve improving the SVP subroutine, is terminating BKZ early, before the algorithm says we are done. It seems that in practice, the early stages of the algorithm lead to the biggest improvements in the quality of the output basis, and so terminating the algorithm early usually gives a basis that is close to BKZ-reduced. For details, see, e.g., the paper of Hanrot, Pujol and Stehlé [HPS2].

4.5 Relations between basis reduction algorithms

To summarize what we discussed in this section, let us give a basic schematic overview of the different basis reduction algorithms, and their relations. Figure 4 shows the four main algorithms discussed in this section, and how they can be obtained from one another. An arrow from one algorithm to another indicates that the first algorithm is a special case of the second (or, equivalently, the second is a generalization of the first). Choosing the blocksize as $k = 2$ reduces KZ to Gauss and BKZ to LLL, while choosing the dimension as $d = k$ and the "slack" parameter as $\delta = 1$ reduces LLL to Gauss, and BKZ to KZ.

5 Solving the Exact Shortest Vector Problem

Besides approximation algorithms that find a reasonably short vector, there also exist exact algorithms that find a *shortest* vector in any lattice. Since finding the shortest vector is NP-hard, one does not expect to

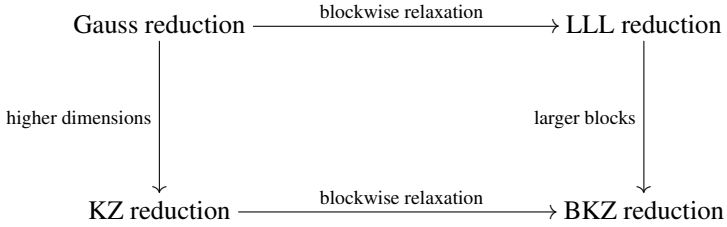

Figure 4: A schematic overview of the basis reduction algorithms discussed in this section.

find polynomial time algorithms, but algorithms with a runtime (super)exponential in the dimension d. But for not too high dimensions d, computationally this may just be within reach. Besides, we can also use these algorithms in lower dimensions as subroutines for the BKZ algorithm discussed in Section 4.4: with a reasonably fast SVP oracle, we can then use the BKZ algorithm to find short vectors in high dimensional lattices.

For finding a shortest vector in a lattice, several techniques are known. In this section, we will consider the two techniques which currently seem the most promising. In Section 5.1, we look at the most natural approach to this problem, which is enumerating all possible short combinations of basis vectors. By considering all combinations up to a fixed length, the shortest one found is guaranteed to be the shortest vector in the lattice. Even though this technique is superexponential, currently it seems to outperform other techniques.

In Section 5.2, we consider a Monte Carlo-style approach to this problem, which consists of making a huge list of short vectors in the lattice. The lattice vectors in this list cover a large portion of the lattice inside a certain ball. From an old result about sphere packings, it follows that this list cannot grow too large, which ultimately results in a high probability that we will eventually find a shortest vector. This algorithm only has an exponential time complexity, which means that eventually (i.e., for sufficiently high dimensions) it will be faster than enumeration. But due to the large constants, the hidden polynomial factors and the exponential space complexity, and the good performance of heuristic enumeration variants, it still has a way to go to be able to compete with current enumeration techniques.

For more details on algorithms for solving the exact shortest vector problem, including an explanation of a third technique of Micciancio and Voulgaris based on Voronoi cells [MV1], see the excellent survey article of Hanrot, Pujol and Stehlé [HPS1].

5.1 Enumeration

The idea of enumeration dates back to Pohst [Poh], Kannan [Ka] and Fincke-Pohst [FP]. It consists of trying all possible combinations of the basis vectors and noting which vector is the shortest. Since "all possible combinations" means an infinite number of vectors, we need to bound this quantity somehow. Take a lattice L with basis $\{\mathbf{b}_1,\ldots,\mathbf{b}_d\}$. Let $R>0$ be a bound such that $\lambda_1(L)\leq R$, e.g. $R=\|\mathbf{b}_1\|$. We would like to be able to enumerate all lattice vectors of norm less than R. As with basis reduction, this enumeration relies on the Gram-Schmidt orthogonalization of the lattice basis. Let $\{\mathbf{b}_1^*,\ldots,\mathbf{b}_d^*\}$ be the GSO-vectors of our basis. Now let $\mathbf{u}\in L$ be any vector of L such that $\lambda_1(L)\leq\|\mathbf{u}\|\leq R$. Recall that every basis vector $\mathbf{b}_i$ can be written as a sum of Gram-Schmidt vectors:

$$\mathbf{b}_i=\mathbf{b}_i^*+\sum_{j=1}^{i-1}\mu_{ij}\mathbf{b}_j^*.$$

Now, using this and the fact that $\mathbf{u}$ is a lattice vector, it is possible to write

$$\mathbf{u} = \sum_{i=1}^{d} u_i \mathbf{b}_i = \sum_{i=1}^{d} u_i \left(\mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^* \right) = \sum_{j=1}^{d} \left(u_j + \sum_{i=j+1}^{d} u_i \mu_{ij} \right) \mathbf{b}_j^*.$$

Representing the lattice vector $\mathbf{u}$ as a sum of Gram-Schmidt vectors allows for a simple representation of projections of $\mathbf{u}$ as well:

$$\pi_k(\mathbf{u}) = \pi_k \left(\sum_{j=1}^{d} \left(u_j + \sum_{i=j+1}^{d} u_i \mu_{ij} \right) \mathbf{b}_j^* \right) = \sum_{j=k}^{d} \left(u_j + \sum_{i=j+1}^{d} u_i \mu_{ij} \right) \mathbf{b}_j^*.$$

Furthermore, since the Gram-Schmidt vectors are by construction orthogonal, the squared norms of $\mathbf{u}$ and its projections are given by

$$\|\pi_k(\mathbf{u})\|^2 = \left\| \sum_{j=k}^{d} \left(u_j + \sum_{i=j+1}^{d} u_i \mu_{ij} \right) \mathbf{b}_j^* \right\|^2 = \sum_{j=k}^{d} \left(u_j + \sum_{i=j+1}^{d} u_i \mu_{ij} \right)^2 \|\mathbf{b}_j^*\|^2. \tag{5.1}$$

We will use (5.1) to bound the number of vectors that need to be enumerated until a shortest vector is found. Recall that the bound R was chosen such that $\|\mathbf{u}\| \leq R$. Since the projection of a vector cannot be longer than the vector itself, it follows that

$$\|\pi_d(\mathbf{u})\|^2 \leq \|\pi_{d-1}(\mathbf{u})\|^2 \leq \ldots \leq \|\pi_1(\mathbf{u})\|^2 = \|\mathbf{u}\|^2 \leq R^2. \tag{5.2}$$

Combining (5.1) and (5.2) gives d inequalities of the form

$$\sum_{j=k}^{d} \left(u_j + \sum_{i=j+1}^{d} u_i \mu_{ij} \right)^2 \|\mathbf{b}_j^*\|^2 \leq R^2, \tag{5.3}$$

for $k = 1, \ldots, d$.

The enumeration now works as follows. First, use (5.3) to enumerate all vectors $\mathbf{x}$ in $\pi_d(L)$ of norm at most R. Then, for each vector $\mathbf{x}$, enumerate all vectors in $\pi_{d-1}(L)$ of norm at most R that project to $\mathbf{x}$ by adding the appropriate multiple of $\mathbf{b}_{d-1}^*$. Repeat this process to enumerate all vectors in $\pi_{d-2}(L)$ and continue down the sequence of projected lattices until all vectors in $\pi_1(L) = L$ have been enumerated.

Thinking of this enumeration in terms of inequalities, (5.3) can be used to give bounds for the unknowns $u_d, \ldots, u_1$, in that order. The first inequality is given by

$$u_d^2 \|\mathbf{b}_d^*\|^2 = \|\pi_d(\mathbf{u})\|^2 \leq R^2.$$

Thus, it follows that $-R/\|\mathbf{b}_d^*\| \leq u_d \leq R/\|\mathbf{b}_d^*\|$. Now, for any fixed $u_d = u_d'$ in this interval, the next inequality becomes

$$(u_{d-1} + u_d' \mu_{d,d-1})^2 \|\mathbf{b}_{d-1}^*\|^2 + u_d'^2 \|\mathbf{b}_d^*\|^2 = \|\pi_{d-1}(\mathbf{u})\|^2 \leq R^2.$$

This inequality can be rewritten as

$$(u_{d-1} + u_d' \mu_{d,d-1})^2 \leq \frac{R^2 - u_d'^2 \|\mathbf{b}_d^*\|^2}{\|\mathbf{b}_{d-1}\|^2}.$$

Taking the square root on both sides shows that u_{d-1} must lie in the interval

$$-u_d' \mu_{d,d-1} - \frac{\sqrt{R^2 - u_d'^2 \|\mathbf{b}_d^*\|^2}}{\|\mathbf{b}_{d-1}\|} \leq u_{d-1} \leq -u_d' \mu_{d,d-1} + \frac{\sqrt{R^2 - u_d'^2 \|\mathbf{b}_d^*\|^2}}{\|\mathbf{b}_{d-1}\|}.$$

Repeating this process leads to an iterative method to derive the interval of u_k once $u_{k+1}, \ldots, u_d$ are fixed. To see this, rewrite (5.3) as

$$\left(u_k + \sum_{i=k+1}^{d} u_i \mu_{ik}\right)^2 \leq \frac{R^2 - \sum_{j=k+1}^{d} \left(u_j + \sum_{i=j+1}^{d} \mu_{ij} u_i\right)^2 \|\mathbf{b}_j^*\|^2}{\|\mathbf{b}_k^*\|^2}.$$

Thus, for fixed $u_{k+1} = u'_{k+1}, \ldots, u_d = u'_d$, u_k must be in the interval

$$-\sum_{i=k+1}^{d} u'_i \mu_{ik} - K \leq u_k \leq -\sum_{i=k+1}^{d} u'_i \mu_{ik} + K,$$

where

$$K = \frac{\sqrt{R^2 - \sum_{j=k+1}^{d} \left(u'_j + \sum_{i=j+1}^{d} \mu_{ij} u'_i\right)^2 \|\mathbf{b}_j^*\|^2}}{\|\mathbf{b}_k^*\|}.$$

Note that it is possible that the interval for u_k is empty (or does not contain integers) for fixed $u'_{k+1}, \ldots, u'_d$.

By trying all possible combinations of $u_1, \ldots, u_d$ that satisfy the inequalities from (5.3), we obtain all lattice vectors $\sum_i u_i \mathbf{b}_i$ of norm smaller than R. Thus, by keeping track of the shortest vector so far, the result will be a shortest vector in the lattice.

It is perhaps simpler to view the enumeration vectors as a search through a tree where each node corresponds to some vector. The i'th level of the tree (where the 0th level is the root) consists of all vectors of $\pi_{d-i+1}(L)$, for $0 \leq i \leq d$. Let $\mathbf{v}$ be a node on the i'th level of the tree, i.e., $\mathbf{v} \in \pi_{d-i+1}(L)$. Then, its children consist of all vectors $\mathbf{u} \in \pi_{d-i+2}(L)$ that get projected onto $\mathbf{v}$ when applying π_{d-i+1}, i.e., $\mathbf{v} = \pi_{d-i+1}(\mathbf{u})$. Thus, the root of the tree consists of $\pi_{d+1}(L) = \{\mathbf{0}\}$, the zero vector. The first level of the tree consists of all vectors in $\pi_d(L) = L(\mathbf{b}_d^*)$ of norm at most R, i.e., all multiples of $\mathbf{b}_d^*$ of norm at most R. The second level of the tree consists of the children of nodes on the first level. This continues until level d, which contains all vectors of $\pi_1(L) = L$ of norm at most R.

Figure 5 depicts a part of the first two levels of such an enumeration tree. Each block consists of a node containing a vector. The first level contains all integer multiples $\lambda_d \mathbf{b}_d^*$ such that $-R/\|\mathbf{b}_d^*\| \leq \lambda_d \leq R/\|\mathbf{b}_d^*\|$. On the second level, three children of $\mathbf{b}_d^*$ are drawn. These correspond to taking $\lambda_d = 1$ in the enumeration and then taking λ_{d-1} in the appropriate interval. If a vector $\mathbf{v}$ is equal to $\mathbf{v} = \lambda_1 \mathbf{b}_1 + \ldots + \lambda_{d-1} \mathbf{b}_{d-1} + \lambda_d \mathbf{b}_d$, then

$$\pi_{d-1}(\mathbf{v}) = \pi_{d-1}(\lambda_d \mathbf{b}_d) + \pi_{d-1}(\lambda_{d-1} \mathbf{b}_{d-1}) = \lambda_d \mathbf{b}_d^* + (\lambda_d \mu_{d,d-1} - \lambda_{d-1}) \mathbf{b}_{d-1}^*.$$

Note that this is exactly the form of the children of $\mathbf{b}_d^*$ in the figure. The other children of the node corresponding to $\mathbf{b}_d^*$ are omitted, as well as the children of the other nodes. Note that the tree is symmetric, as for each vector $\mathbf{v}$ in the tree, $-\mathbf{v}$ is in the tree as well. During the enumeration, only one side of the tree needs to be explored.

Such enumeration trees grow quite large. In fact, they become exponentially large, dependent on the precision of the bound R. The lower this bound, the smaller the corresponding enumeration tree. Thus, while such methods give an exact solution to the shortest vector problem, their running time is not polynomially bounded. In order to optimize the running time, lattice basis reduction algorithms are often used before enumerating. This improves the GSO of the basis, reduces the numbers μ_{ij} by size-reduction and additionally gives an exponential approximation to the length of a shortest vector (which in turn gives exponential upper bounds for the running time of the enumeration).

Algorithm 7 is a simplified description of the enumeration algorithm. Each node corresponds to a coefficient vector $\mathbf{u} = (u_1, \ldots, u_d)$ which corresponds to a lattice vector $\sum_i u_i \mathbf{b}_i$. The algorithm starts at the coefficient vector $(1, 0, \ldots, 0)$, which corresponds to the lattice vector $\mathbf{b}_1$. If the algorithm cannot go further down in line 3, this means we are at a leaf of the tree that corresponds to a lattice vector with a norm that

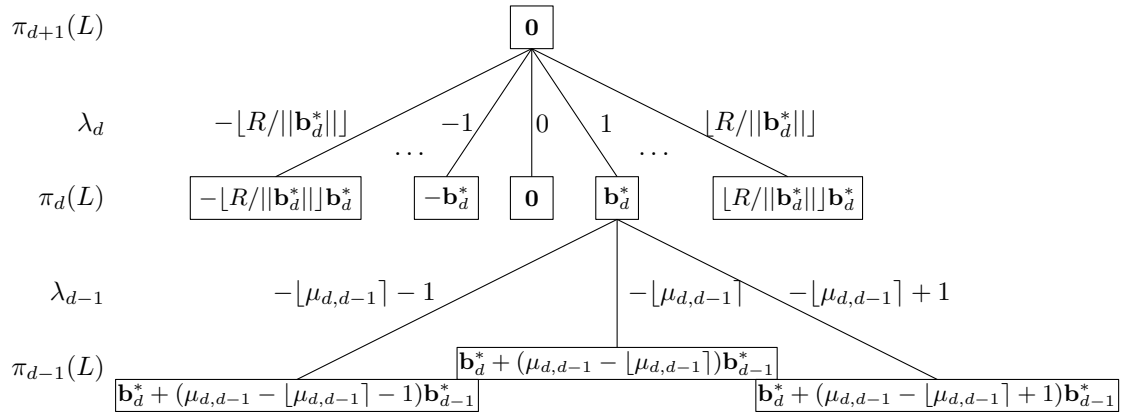

Figure 5: The first two levels of the enumeration tree.

Algorithm 7 An enumeration algorithm

Require: a reduced bases $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L and its Gram-Schmidt coefficients $\mu_{i,j}$ and norms $\|\mathbf{b}_i^*\|$
Ensure: the output lattice vector $\sum_i u_i \mathbf{b}_i \in L$ is a shortest vector of L

1: **repeat**
2: **if** norm of current node is below the bound **then**
3: go down a level, to the child with minimum norm
4: **else**
5: go up a level, to the unvisited sibling of the parent with smallest norm
6: **end if**
7: **until** all nodes have been searched

is smaller than the bound. Thus, we have found a new candidate for a shortest vector and can update our bound. We also store the coefficient vector of the shortest vector found so far. In line 5 of the algorithm we go back up the tree, which means that we have finished searching a particular subtree. Instead of going to the parent of the current node, we go to its sibling that we have not yet visited with minimal norm. If the algorithm cannot go up a level in line 5, that means we are in the root and have searched all nodes in the tree.

5.1.1 Improved enumeration techniques

Schnorr and Hörner proposed a technique to speed up the enumeration algorithm based on *pruning* [SH]. Consider the enumeration algorithm as the depth-first search of a tree. In pruned enumeration, so-called branches or subtrees are excluded from the search (pruned from the tree). This means that part of the tree is ignored and will not be searched for a shortest vector. Now, for Schnorr-Hörner pruning, these subtrees are chosen in such a way that the probability that a shortest vector is inside that subtree is too small compared to some threshold. This means that with some small probability, no shortest vector is found. However, the vector that is found instead is still reasonably short and depending on the thresholds, the running time should decrease enough to justify this error probability.

Recently, Gama, Nguyen and Regev introduced a new concept called *extreme pruning* [GNR]. Their main idea is to prune a large number of branches, which significantly reduces the search tree. The probability that a shortest vector is found becomes very low as a result. However, the running time of the enumeration is reduced by a much bigger factor. Therefore, the pruned enumeration can be executed several times so that a shortest vector is found with much higher probability. As the running time of a single enumeration is reduced significantly, this results in a net decrease in running time when performing several enumerations. The analysis by Gama et al. shows that extreme pruning decreases the running time by an exponential

factor and their paper contains experimental results to back this up. Using extreme pruning, they were able to find a shortest vector in a hard knapsack lattice of dimension 110.

Gama, Nguyen and Regev also introduce so-called *bounding functions* to improve the analysis of pruned enumeration methods, and show that the analysis of the original method by Schnorr and Hörner was not optimal. Additionally, Schnorr recently introduced new enumeration methods [Sno4], using results of Gama, Nguyen and Regev [GNR].

5.2 Sieving

While enumeration seems the most natural way to find a shortest vector in a lattice, it may not be the most efficient way. Recently so-called sieving algorithms have been studied, achieving exponential running times instead of the superexponential runtime of enumeration algorithms. These algorithms require exponential space as well, and even though asymptotically these algorithms are faster, in low dimensions the small constants of the enumeration algorithms seem to outweigh their extra logarithmic factor in the exponent. But let us not be too pessimistic: certainly in sufficiently high-dimensional lattices, these sieving algorithms outperform enumeration algorithms with respect to the time needed to find a shortest vector.

Below we will sketch the sieving algorithm of Micciancio and Voulgaris [MV2], which uses ideas from Ajtai et al. [AKS], and was further studied by Schneider [Sne1, Sne2] and Pujol and Stehlé [PS2]. First of all, the length of a shortest vector is generally not known, but for now we assume that some estimate $\mu \approx \lambda_1(L)$ is known. If it turns out that there exist no lattice vectors of length at most μ, we can always increase μ, while if μ is too large, we can just decrease μ and run the algorithm again until we find a shortest vector. Then, for many iterations, the algorithm first samples short 'error' vectors $\mathbf{e}_i$, calculates an associated lattice vector $\mathbf{v}_i$, 'reduces' the vector $\mathbf{r}_i = \mathbf{v}_i + \mathbf{e}_i$ to a shorter vector $\mathbf{r}'_i = \mathbf{v}'_i + \mathbf{e}_i$ using the lattice vectors that are already in the list Q, and then adds the lattice vector $\mathbf{v}'_i$ to the list Q. Repeating this, the algorithm tries to exhaust the space of short lattice vectors by adding more and more short lattice vectors (with norm at most $\|\mathbf{B}\|$, and pairwise at least μ apart) to a long list Q, until either two of the lattice vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ in Q are at most μ apart or (which will almost never happen if $\mu > \lambda_1(L)$) we have used too many samples.

Algorithm 8 The Micciancio-Voulgaris sieving algorithm

Require: a reduced basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$ of L, and a value $\mu \in \mathbb{R}$
Ensure: if $\mu > \lambda_1(L)$, then with high probability the algorithm finds a vector $\mathbf{v} \in L$ with $\|\mathbf{v}\| \leq \mu$

$Q \leftarrow \emptyset$
$\xi \leftarrow 0.685$
$N \leftarrow \text{poly}(d) \cdot 2^{3.199d}$
for $i = 1$ to N **do**
  $\mathbf{e}_i \in_R B_d(\mathbf{0}, \xi\mu)$
  $\mathbf{r}_i \leftarrow \mathbf{e}_i \mod B$
  while $\exists \mathbf{v}_j \in Q : \|\mathbf{r}_i - \mathbf{v}_j\| \leq (1 - \frac{1}{d})\|\mathbf{v}_i\|$ **do**
    $\mathbf{r}_i \leftarrow \mathbf{r}_i - \mathbf{v}_j$
  end while
  $\mathbf{v}_i \leftarrow \mathbf{r}_i - \mathbf{e}_i$
  if $\mathbf{v}_i \notin Q$ **then**
    if $\exists \mathbf{v}_j \in Q : \|\mathbf{v}_i - \mathbf{v}_j\| < \mu$ **then**
      return $\mathbf{v}_i - \mathbf{v}_j$
    end if
    $Q \leftarrow Q \cup \{\mathbf{v}_i\}$
  end if
end for
return $\perp$

If we find two vectors $\mathbf{v}_i, \mathbf{v}_j \in L$ which are at most μ apart, then the vector $\mathbf{v} = \mathbf{v}_i - \mathbf{v}_j$ is also in the

lattice and has a length of at most μ, and the algorithm returns $\mathbf{v}$ as a solution. If we never find two such vectors, then all vectors in the list are always pairwise at least μ apart. But since the sampled vectors have a bounded length, we can prove a rigorous upper bound on the maximum size of the list Q at any point in time. Furthermore, we can prove that at each point in time, with sufficiently high (fixed) probability, either new vectors are added to the list or a solution is found. This means that unless the algorithm terminates by finding a vector of length at most μ, with sufficiently high probability new vectors keep getting added to the list, while the list can never grow larger than a fixed constant. Adding these facts together guarantees that with high probability, we must escape the loop somewhere by finding a short lattice vector.

Above we made some claims which are far from obvious, and we will motivate them now. First, we will investigate why the list size is bounded by an (exponential) constant. For this we use an old result of Kabatiansky and Levenshtein, given below.

Theorem 5.1. *[KL, Consequence 1] Let $\phi_0 \approx 1.100 > \frac{\pi}{3}$ be a root of $\sin x + \tan x = \ln\left(\frac{1+\sin x}{1-\sin x}\right)$. Let Q be any set of points in $\mathbb{R}^d$ such that the angle between any two points $\mathbf{q}_1, \mathbf{q}_2 \in Q$ is at least $\phi < \phi_0$. Then*

$$|Q| \leq 2^{\left(-\frac{1}{2}\log_2(1-\cos(\phi))-0.099\right)d}.$$

In particular, for $\phi = \frac{\pi}{3}$ we have

$$|Q| \leq 2^{0.401d}.$$

To be able to apply this result to the list Q from the algorithm, we first divide Q into so-called spherical shells or coronas. Let $Q_0 = \{\mathbf{v} \in Q : 2\|\mathbf{v}\| \leq \mu\}$ and $\gamma = 1 + \frac{1}{d}$. Let Q_k, for k ranging from 1 to $\log_\gamma \|\mathbf{B}\|$, be defined as

$$Q_k = \{\mathbf{v} \in Q : \gamma^{i-1}\mu < 2\|\mathbf{v}\| \leq \gamma^i \mu\}.$$

Since $\|\mathbf{v}_i - \mathbf{v}_j\| > \mu$ for any two vectors in the list, and since $\|\mathbf{v}_i - \mathbf{v}_j\| > (1 - \frac{1}{d})\|\mathbf{v}_i\|$ if $\mathbf{v}_i$ was added to the list later than $\mathbf{v}_j$, one can show that the angle between any two vectors in Q_k is large, and bounded from below by an angle less than $\pi/3$:

$$\phi_{\mathbf{v}_1,\mathbf{v}_2} = \cos^{-1}\left(\frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\|\|\mathbf{v}_2\|}\right) \geq \cos^{-1}\left(1 - \frac{1}{2}(\xi + \sqrt{\xi^2+1})^{-2} + o(1)\right) = \phi_0 < \frac{\pi}{3}.$$

This then gives us bounds on the sizes of each $\tilde{Q}_k$, which are stronger than the 0.401 for $\phi_0 = \frac{\pi}{3}$, namely $|Q_k| \leq 2^{(0.401+\log_2(\xi+\sqrt{\xi^2+1}))d}$. Since the number of spherical shells is at most $\log_\gamma \|\mathbf{B}\| = \text{poly}(d)$, which for $\gamma = 1 + \frac{1}{d}$ is at most polynomial in d, the total list size is bounded by

$$|Q| = \sum_{k=0}^{\log_\gamma \|\mathbf{B}\|} |Q_k| \leq \text{poly}(d) \cdot 2^{(0.401+\log_2(\xi+\sqrt{\xi^2+1}))d}.$$

For the constant $\xi = 0.685$, as chosen by Micciancio and Voulgaris [MV2], this leads to

$$|Q| \leq \text{poly}(d) \cdot 2^{1.325d}.$$

This proves an exponential upper bound on the maximum length of the list Q, as needed.

The only thing left to prove or sketch is the fact that collisions (ending up with a reduced vector $\mathbf{r}_i = \mathbf{v}_i + \mathbf{e}_i$ such that $\mathbf{v}_i$ is already in the list Q) cannot occur too often, i.e. with a sufficiently high fixed probability we will not get collisions. Then, if the number of iterations is large enough, with high probability we will add many vectors to the list, while the list will never be too long, proving we must find a shortest vector at some point. For this, we use the following result.

Lemma 5.2. *Let L be a lattice, and let $\mu \geq \lambda_1(L)$. Let $\mathbf{s}$ be a vector in the lattice with $\|\mathbf{s}\| \leq \mu$. Let $I = B_d(\mathbf{0}, \xi\mu) \cap B_d(\mathbf{s}, \xi\mu)$. Then the probability that the error vector $\mathbf{e}_i$, which is sampled uniformly at random from $B_d(\mathbf{0}, \xi\mu)$, is inside I is at least*

$$P(\mathbf{e}_i \in I) \geq \frac{1}{\text{poly}(d) \cdot 2^{\frac{1}{2}\log_2\left(\xi^2/(\xi^2-\frac{1}{4})\right)d}}.$$

In particular, for $\xi = 0.685$, this leads to

$$P(\mathbf{e}_i \in I) \geq \frac{1}{\text{poly}(d) \cdot 2^{0.5489d}}.$$

The argument used to prove this is proving an upper bound on the ratio between the volume of this region I and the volume of the entire hypersphere $B_n(\mathbf{0}, \xi\mu)$. The region I is particularly interesting, because we can show that if $\mathbf{e}_i \in I$, then the probability that the resulting lattice vector $\mathbf{v}_i$ is already in the list Q is not larger than the probability of finding a lattice vector $\mathbf{v}_i$ that leads to a solution, i.e. with $\|\mathbf{v}_i - \mathbf{v}_j\| \leq \mu$ for some $\mathbf{v}_j \in Q$. So the probability of getting a collision, conditioned on the fact that $\mathbf{e}_i$ is sampled from I, is at most $\frac{1}{2}$. This then implies that the total probability of not getting a collision is bounded from below by

$$P(\mathbf{v}_i \notin \{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}\}) \geq \underbrace{P(\mathbf{v}_i \notin \{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}\} \mid \mathbf{e}_i \in I)}_{\geq \frac{1}{2}} \cdot P(\mathbf{e}_i \in I) \geq \frac{1}{\text{poly}(d) \cdot 2^{0.5489d}}.$$

So the probability of not getting a collision is exponentially small in d, but does not depend on the index i. So after $2N/p$ samples, we expect to get at least $2N/p \cdot p = 2N$ non-collisions, and using e.g. Chernoff's bound, one can show that with probability exponentially close to 1, we get at least N non-collisions. But the list is never longer than N, which means that the probability of failure is exponentially small in the dimension d.

Theorem 5.3. *Let L be a lattice, and let $\mu \geq \lambda_1(L)$. Then in time at most $\text{poly}(d) \cdot 2^{3.199d}$ and space at most $\text{poly}(d) \cdot 2^{1.325d}$, with high probability Algorithm 8 finds a vector $\mathbf{s} \in L$ with $\|\mathbf{s}\| \leq \mu$.*

So with only exponential time and space complexity, one can find a shortest vector in a lattice. This means that for sufficiently large d, sieving will be faster than enumeration, which is superexponential in the dimension d. However, enumeration only has an extra logarithmic factor in the exponent, and it takes quite a while before this factor overtakes the larger constants of sieving. Moreover, for huge d, both algorithms will fail to produce a shortest vector in a reasonable amount of time anyway. Still, it is an interesting question to find out where exactly this crossover point is, where sieving starts becoming faster than enumeration.

5.2.1 Improved sieving techniques

For the sieving algorithm described above, several theoretical and practical improvements were suggested. Pujol and Stehlé [PS2] proposed a variant of the sieving algorithm described above, exploiting a birthday attack to further reduce the constant in the exponent. This lead to the following result.

Theorem 5.4. *[PS2, Theorem 1] Let L be a lattice, and let $\mu \geq \lambda_1(L)$. Then in time at most $\text{poly}(d) \cdot 2^{2.465d}$ and space at most $\text{poly}(d) \cdot 2^{1.233d}$, with probability exponentially close to 1 the sieving algorithm of Pujol and Stehlé [PS2, Fig. 2] finds a vector $\mathbf{s} \in L$ with $\|\mathbf{s}\| \leq \mu$.*

Still, this seems to be far off the actual practical runtime, which is much better than $2^{2.465d}$. An interesting open problem is whether one can prove that a better runtime can be achieved for any lattice.

Instead of provable variants, several papers have also investigated heuristic improvements without a provable runtime, but which seem to work faster in practice. Micciancio and Voulgaris suggested an improvement to their sieving algorithm, which they called the *GaussSieve* algorithm. Instead of only reducing new vectors with old vectors from the list, one then also reduces the vectors that are already in the list with the new vector. This makes sure that each pair of vectors in the list is Lagrange reduced, or Gaussian reduced. This then implies that the angle between any two vectors is at least $\pi/3$. So the space complexity is then provably bounded from above by $2^{0.401d}$.

Besides upper bounds on the list size, an old result from Shannon also gives a lower bound, depending on the minimum angle ϕ_0.

Theorem 5.5. *[Sha, Equation (21)] Let Q^* be the largest set of points in $\mathbb{R}^d$ such that the angle between any two points $\mathbf{q}_1, \mathbf{q}_2 \in \tilde{Q}$ is at least ϕ_0. Then, for large d,*

$$|Q^*| \geq 2^{-\log_2(\sin\phi_0)d+o(d)}.$$

In particular, for $\phi_0 = \frac{\pi}{3}$ we have

$$|Q^*| \geq 2^{0.208d+o(d)}.$$

Although this is a lower bound, this result has been around for quite a while, and it seems that for (almost) all lattices, this lower bound is actually an upper bound: most lattices have lower densities than $2^{-\log_2(\sin\phi_0)d+o(d)}$. So in practice, using the heuristic *GaussSieve*, we do not expect to have a list of length $2^{0.401d}$, but only about $2^{0.208d}$.

However, while the list size is bounded sharper from above theoretically and heuristically, Micciancio and Voulgaris were not able to prove an exponential runtime. For their original algorithm, they could prove that the probability of collisions is sufficiently small, i.e., sufficiently far from 1. With the *GaussSieve* algorithm, this proof no longer works, and so theoretically it is not clear whether the probability of collisions can be bounded away from 1. Experimentally however, the algorithm seems to perform quite well, mainly due to the fact that the reduction of vectors happens far from as often as the worst-case predicts. For more details on these experiments, see Schneider [Sne1].

Finally, a somewhat different direction in sieving was initiated by Ajtai et al. [AKS]. Instead of exhausting the space of short vectors with one huge list of vectors, one starts with a list of longer vectors, and reduces the size of the list and the norms of the vectors in these lists over several iterations. One could argue this fits the term 'sieving' better, as then a sieve is actually applied. Further improvements in this direction were studied by Nguyen and Vidick [NVi] and Wang et al. [WLTB]. Note that, even though these algorithms may perform quite well in practice, the worst-case upper bounds of Pujol and Stehlé are the best known so far.

6 Measuring the Practical Security of Lattice-Based Cryptosystems

In the previous sections we described the different techniques that are used when solving lattice problems. Now we would like to know how effective these techniques are in practice, when breaking cryptosystems based on such problems. In this section, we will consider the question: How much effort is required of an attacker to break lattice-based cryptosystems using these techniques?

Several problems arise when trying to answer this question. First of all, there are multiple ways to solve lattice problems. Which method should an attacker use to minimize his effort in breaking systems based on such problems? Secondly, the behavior of basis reduction algorithms is not always well understood. Even before lattice problems were used to build cryptosystems, basis reduction algorithms were used in cryptanalysis to break knapsack-based cryptosystems. The LLL algorithm seemed to solve the Shortest Vector Problem that arose from these knapsack lattices every time, despite the fact that it is a hard problem. Finally, not much is known about the running time of BKZ. The bounds that we know to hold in theory

do not seem to be tight in practice. In fact, in practice BKZ appears to outperform other basis reduction algorithms, even those that have good bounds on their time complexity.

In this section we will first look at typical ways that lattice problems are used in cryptography. We will also consider how to use the different techniques to solve these problems. Then, we take a look at work by Gama and Nguyen that considers the practical performance of basis reduction algorithms. However, their work was not specifically aimed at cryptography, so we will also consider work by Rückert and Schneider, who adapted the approach by Gama and Nguyen to the analysis of breaking lattice-based cryptosystems. In order to create a framework encompassing cryptosystems based on both the LWE and the SIS problems, Rückert and Schneider assume the best way to solve LWE is by reducing it to SIS and applying basis reduction. Some work by Lindner and Peikert, which we discuss afterwards, suggests that this might not be the optimal strategy for an attacker. Finally, we look at the new results of Chen and Nguyen, who introduce BKZ 2.0 and try to combine theoretical and experimental approaches to analyze its performance in practice.

6.1 Lattices and cryptography

In Section 3 we discussed several lattice problems. Here, we will consider how these problems appear in lattice-based cryptography. We will consider GGH-type cryptosystems and collision-resistant hash functions based on lattices.

The GGH-cryptosystem was named for its creators Goldreich, Goldwasser and Halevi [GGH2] and was one of the first cryptosystems based on lattice problems. The concept is quite simple. The private key is a 'good' basis of the lattice, while the public key is a 'bad' basis of the lattice. To encrypt a message, first encode it into a lattice point using the public basis. Then, draw a small error vector from an error distribution and add it to the encoded lattice point. The ciphertext is the resulting vector. To decrypt a ciphertext, use Babai's rounding method with the private basis to find the closest lattice point. Decode this lattice point to retrieve the message. The same idea can be used to digitally sign messages, by encoding the message in any vector and using the private basis to find a close lattice vector, which will be the signature. Because the error vector cannot be too big (this would lead to decryption errors) the security of such systems is related to the BDD problem.

Goldreich, Goldwasser and Halevi also described how to construct collision-resistant hash functions from lattice problems [GGH1]. Recall the description of the modular lattice $L_{\mathbf{A},q}$ from Section 3, as well as the associated SIS problem. Now consider the hash function given by

$$h_{\mathbf{A}} = \mathbf{A}\mathbf{x} \bmod q, \quad \text{for } \mathbf{x} \in \{0,1\}^m.$$

Say someone finds a collision, i.e., two 0,1-vectors $\mathbf{x}_1 \neq \mathbf{x}_2$ such that $\mathbf{A}\mathbf{x}_1 = \mathbf{A}\mathbf{x}_2 \bmod q$. Then, $(\mathbf{x}_1 - \mathbf{x}_2) \in \{-2,-1,0,1,2\}^m$ is a solution for the SIS problem, as $\mathbf{A}(\mathbf{x}_1 - \mathbf{x}_2) = 0 \bmod q$. Thus, finding collisions is as hard as finding a solution to the SIS problem in the lattice $L_{\mathbf{A},q}$.

These two examples that lattice-based constructions of cryptographic primitives are often closely related to the associated lattice problems. Thus, solving these lattice problems seems like the most straightforward way to break such systems. In the remainder of this section we shall see how the techniques described in the previous sections fare against lattice problems in practice.

6.2 Basis reduction in practice

Prompted by the unexpectedly good behavior of basis reduction algorithms, Gama and Nguyen analyzed the performance of such algorithms in practice. They experimented with several reduction algorithms on a variety of lattices. Their goal was to show what basis reduction algorithms are capable of, and to use these results to compare the practical difficulty of the HSVP, USVP and approximate SVP problems, which were described in Section 3. Here, we will briefly describe their methods and then discuss their results.

First, we take a look at their method of lattice generation. Then, we will discuss the basis reduction algorithms that they used in their experiments. Finally, we consider the quality of the bases that resulted from the experiments, as well as the running time of the experiments.

6.2.1 Generating lattices

If we want to determine the performance of basis reduction algorithms in practice, we need input lattices for our experiments. Thus, the first issue that arises is how we generate these lattices. For examining the average case performance of the algorithms on HSVP and approximate SVP, Gama and Nguyen took a large sample of lattices from a distribution due to Goldstein and Mayer [GM]. These lattices have the property that the successive minima of the lattice satisfy

$$\lambda_i(L) \approx \sigma(L) = \sqrt{\frac{d}{2\pi \mathrm{e}}}(\mathrm{vol}(L))^{1/d},$$

for all $1 \leq i \leq d$. This means that all minima are close to the expected shortest length $\sigma(L)$ which follows from the Gaussian heuristic, as explained in Section 2.

However, the unique shortest vector problem requires extra structure in the lattice, which makes them quite different from Goldstein–Mayer lattices. The lattice gap $\lambda_2(L)/\lambda_1(L)$ needs to exceed a value γ, whereas in Goldstein–Mayer lattices all successive minima are close to each other. Unfortunately, there is no 'standard' way to construct lattices with a prescribed gap. Therefore, Gama and Nguyen chose to work with two classes of lattices where they could pick the approximate lengths $\lambda_1(L)$ and $\lambda_2(L)$ and then they used orthogonality to ensure that the appropriate vectors have these lengths. It is not entirely clear how these choices affect the performance of basis reduction. The basis reduction algorithms may be able to exploit the orthogonality in order to perform better. Therefore, aside from the two aforementioned classes, they also perform experiments on lattices that arise from knapsack problems. However, there is no known formula for the second minimum λ_2 of such lattices. As a result, the second minimum needs to be approximated heuristically in order to prescribe the gap.

Once these random lattices are generated, they need to be represented by means of a basis. A given basis might have special properties that influence the performance of a basis reduction algorithm. Gama and Nguyen chose to work with random bases in order to remove such influences. However, there is no standard notion of random bases for a given lattice. Gama and Nguyen define a random basis as a basis consisting of relatively large lattice vectors that are chosen using a randomized heuristic. They do not explicitly give their methods to randomly generate lattice bases, but they refer to the description of the GGH-cryptosystem [GGH2], which details heuristics to randomize a lattice basis. In the experiments, they performed the basis reduction on at least twenty randomized bases for each lattice. In order to prevent the reduction algorithms from taking advantage of special properties of the original basis, the randomization must ensure that the randomized basis vectors are not short.

6.2.2 Algorithms

Basis reduction algorithms give an approximate solution to the different variants of the shortest vector problem. They produce bases $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ such that the first basis vector $\mathbf{b}_1$ is relatively short. To measure the quality of this short vector, the *approximation factor* is defined as $\|\mathbf{b}_1\|/\lambda_1(L)$ and the *Hermite factor* is defined as $\|\mathbf{b}_1\|/\mathrm{vol}(L)^{1/d}$, where d is the lattice rank. These factors try to capture the notion of the "shortness" of the vector $\mathbf{b}_1$ in regards to the approximate SVP and HSVP problems, respectively.

As mentioned in Section 5, there also exist algorithms that solve SVP exactly. The algorithms mentioned in Section 5.1 perform an exhaustive search based on enumeration methods and have a time complexity that is at least exponential in the lattice rank. Gama and Nguyen pose that these algorithms cannot be run on lattices where the rank is greater than 100. For such ranks, only approximation algorithms such as basis reduction algorithms are practical. However, this observation was made before the introduction of extreme

pruning by Gama, Nguyen and Regev [GNR]. Extreme pruning has been used to find a shortest vector in lattices of rank 110 (with a running time of 62.12 CPU days). As we will see in the subsection on BKZ 2.0, being able to perform enumeration on lattices of higher dimensions makes the analysis of the output quality of BKZ easier.

For their experiments, Gama and Nguyen use three different reduction algorithms: LLL [LLL], DEEP [SE] and BKZ [SE]. These algorithms were described in previous sections.

Gama and Nguyen used the NTL [Sho] (version 5.4.1) implementations of BKZ and DEEP in their experiments. In NTL, both BKZ and DEEP use a 'blocksize' parameter β. For higher values of β, the quality of the reduction increases, but the running time increases as well. In addition to the quality of the reduced bases, Gama and Nguyen examined the running times of BKZ and DEEP in their experiments.

6.2.3 Results

Now, let us consider the results of the experiments of Gama and Nguyen. In the cases of HSVP and approximate SVP, they measure the performance of the basis reduction algorithms by the Hermite and approximation factors of the resulting vector, respectively. In the case of USVP, they only measure the performance by checking whether the algorithm finds one of the two shortest vectors or not.

For each of the three lattice problems, we will examine the theoretical expectations of the performance of the basis reduction algorithms first. Then, we compare these expectations to the experimental results and attempt to explain the difference between theory and practice. Finally, we consider the experimental running time of BKZ and DEEP, as well as the running time of an exhaustive enumeration method.

HSVP Recall the definition of Hermite SVP from Section 3. The goal is to find a vector of norm at most $\gamma \mathrm{vol}(L)^{1/d}$ in a d-rank lattice L for some approximation factor γ. Theoretically, basis reduction algorithms solve HSVP with a Hermite factor $\|\mathbf{b}_1\|/\mathrm{vol}(L)^{1/d} = (1+\varepsilon)^d$, where ε depends on the algorithm and its parameters. For LLL with appropriate reduction parameters, the Hermite factor is theoretically provable to be $\lessapprox (\gamma_2)^{(n-1)/2} = (4/3)^{(n-1)/4} \approx 1.0746^n$, where γ_2 is Hermite's constant for rank 2 lattices.

The DEEP algorithm is based on the LLL algorithm and theoretically it is not known to perform better than LLL. In other words, there is no upper bound known for the Hermite factor of DEEP, except the upper bound of LLL. However, DEEP is expected to perform better than LLL in practice. In contrast, BKZ does have better theoretical upper bounds. By using arguments similar to those used by Schnorr [Sno1], it can be proven that the Hermite factor of BKZ is at most $\sqrt{\gamma_\beta}^{1+(d-1)/(\beta-1)}$, where β is the blocksize parameter. For $\beta = 20$ the upper bound on the Hermite factor is approximately 1.0337^d and for $\beta = 28$ this bound is approximately 1.0282^d. But are these theoretical bounds tight in practice?

The first observation that Gama and Nguyen make from their experiments is that the Hermite factor of the result of basis reduction does not seem to depend on the lattice, in the sense that it does not vary strongly between lattices. Only when the lattice has exceptional structure will the Hermite factor be relatively small compared to the general case. Here, exceptional structure means that either $\lambda_1(L)$ is very small, or the lattice contains a sublattice (of lower rank) of very small volume, i.e., a sublattice spanned by a few relatively short vectors.

The next observation is that, when there is no such exceptional structure in the lattice, the Hermite factor appears to be exponential in the lattice rank. This agrees with the theoretical predictions. However, the specific constants that are involved appear to differ in practice. The experiments show that the Hermite factor is approximately of the form e^{ad+b}, where d is the lattice rank and the constants a and b depend only on the reduction algorithm. Gama and Nguyen are only interested in rough estimations and they simplify e^{ad+b} to δ^d. Table 1 shows the base δ of the average Hermite factors that were derived from the experiments. The value δ is called the root-Hermite factor, as it is the d'th root of the Hermite factor.

Table 1 shows that DEEP and BKZ exhibit the same exponential behavior as LLL, except that their con-

Algorithm-β	LLL	DEEP-50	BKZ-20	BKZ-28
Experimental root-Hermite factor δ	1.0219	1.011	1.0128	1.0109
Theoretical proven upper bound	1.0746	1.0746	1.0337	1.0282

Table 1: Experimental root-Hermite factor compared to theoretical upper bounds.

stants are smaller. The constants for BKZ and DEEP are roughly the square root of the constants for LLL. To give a concrete example what these constants mean in practice, consider lattices of rank $d = 300$. According to these experimental constants, LLL will obtain a vector with Hermite factor of approximately $1.0219^{300} \approx 665$, while the theoretical upper bound is $1.0746^{300} \approx 2958078142$. Furthermore, BKZ-20 will obtain a vector with a Hermite factor of approximately $1.0128^{300} \approx 45$, while the theoretical upper bound is $1.0337^{300} \approx 20814$.

The results of LLL give some good insight into average-case versus worst-case behavior. It is known that in the worst-case, the Hermite factor of LLL is equal to the theoretical upper bound $\gamma_2^{(n-1)/2} = (4/3)^{(n-1)/4}$. This occurs when the input is a lattice basis such that all its 2-rank projected lattices are critical, i.e., they satisfy Hermite's bounds that were mentioned in Section 2. However, this behavior is caused by a worst-case basis and not by a worst-case lattice. The experiments showed that when a random basis of these lattices was reduced instead of this worst-case basis, the resulting Hermite factor was again 1.0219^d.

It is harder to explain the gap between theory and practice for the BKZ algorithm. The BKZ algorithm uses projected lattices of rank β, where β is the blocksize. Although it is known that these projected lattices do not have the same distribution as random lattices of rank β, no good model for their distribution is known. This makes it difficult to analyze the performance of BKZ theoretically. This holds true for the blocksizes considered in the experiments, $\beta \leq 40$. This raised the question whether this behavior also occurs in higher blocksizes.

Based on their results, Gama and Nguyen conclude that the best algorithms can reach a Hermite factor of roughly 1.01^d. They also conclude that solving HSVP for Hermite factor d using BKZ is currently 'easy' for $d \leq 450$, as δ^d is approximately linear in d in this case, e.g., $1.013^{450} \approx 334 \leq 450$. However, they note that a Hermite factor of 1.005^d cannot be reached for rank $d = 500$ in practice, unless the lattice has an exceptional structure as discussed before.

Approximate SVP Recall from Section 3 that any algorithm that solves HSVP with Hermite factor γ can be used to solve approximate SVP with approximation factor γ^2. This leads to the expectation that the reduction algorithms can solve approximate SVP with an approximation factor that is equal to the square of the Hermite factor. Since the experimental results for HSVP show that the Hermite factor is approximately δ^d with δ as in Table 1, it is to be expected that the approximation factor is roughly $(\delta^d)^2 = (\delta^2)^d$. Assuming that the best reduction algorithms can reach a Hermite factor of roughly 1.01^d, it is expected that they will reach an approximation factor of roughly $1.01^{2d} \approx 1.02^d$.

The experiments showed that this expectation was true in the worst-case. Gama and Nguyen constructed lattices where the approximation factor was the square of the Hermite factor. However, on the average it appeared that the approximation factor was in fact roughly the same as the Hermite factor 1.01^d, rather than its square $1.01^{2d} \approx 1.02^d$. A possible explanation is that in lattices where $\lambda_1(L) \geq \text{vol}(L)^{1/d}$, any algorithm that reaches a Hermite factor $\|\mathbf{b}_1\|/\text{vol}(L)^{1/d}$ will reach an approximation factor $\|\mathbf{b}_1(L)\|/\lambda_1(L) \leq \|\mathbf{b}_1(L)\|/\text{vol}(L)^{1/d}$. Thus, approximate SVP can only be harder than HSVP in lattices where $\lambda_1(L) \leq \text{vol}(L)^{1/d}$. However, if $\lambda_1(L)$ becomes too small compared to $\text{vol}(L)^{1/d}$, the lattice will have an exceptional structure. This structure can then be exploited by basis reduction algorithms in order to improve their results.

The worst-case results are based on experiments with LLL, but Gama and Nguyen note that BKZ and DEEP perform essentially the same except for the better constants. They conclude that current algorithms can reach approximation factors of 1.01^d on the average and 1.02^d in the worst case. This suggests that

solving approximate SVP with approximation factor d is easy on the average for $d \leq 500$, because 1.01^d is approximately linear in d for these values of d.

USVP Recall the definition of USVP from Section 3. The goal is to find a shortest vector in a lattice L where the shortest vector $\mathbf{u}$ is unique. Here, unique means that all vectors of length $\leq \gamma\|\mathbf{u}\|$ are a multiple of $\mathbf{u}$ for some gap constant γ. Recall from Section 3 that any algorithm that achieves an approximation factor $\leq \gamma$ can solve the unique shortest vector problem with gap γ. Thus, using the results of the approximate shortest vector problem, the expectation is that USVP can be solved for gap roughly $\geq 1.02^d$, the square of the Hermite factor.

As mentioned in the part about lattice generation, Gama and Nguyen constructed two classes of lattices where they could choose the lattice gap. For these lattices they found that LLL would retrieve the unique shortest vector whenever the gap was exponential in the lattice rank, as predicted. However, the gap did not need to be the square of the Hermite factor, which is approximately 1.042^d for LLL. Instead, LLL obtains the unique shortest vector with high probability as soon as the gap becomes a fraction of the Hermite factor (and not its square). For the first class of lattices this happened whenever $\lambda_2/\lambda_1 \geq 0.26 \cdot 1.021^d$ and for the second class whenever $\lambda_2/\lambda_1 \geq 0.45 \cdot 1.021^d$. For BKZ, the constants were so close to 1 that lattices of rank < 200 did not provide sufficient accuracy on the constants. The results from higher ranks seemed to indicate that BKZ could find the unique shortest vector whenever the gap was greater than $0.18 \cdot 1.012^d$ in the first class.

However, these constructions had an exceptional structure when compared to general USVP-instances, which could affect the performance of reduction algorithms. Thus, Gama and Nguyen repeated their experiments for so-called Lagarias-Odlyzko lattices [LO], which are lattices that arise from knapsack problems. Using heuristical estimates to determine the gap, they found that LLL could retrieve the unique shortest vector whenever the gap was greater than $0.25 \cdot 1.021^d$ and BKZ achieved this whenever the gap was greater than $0.48 \cdot 1.012^d$. This agrees with their earlier result that USVP is easy to solve whenever the gap is a fraction of the Hermite factor, rather than its square.

Experimental running time As mentioned in their description, no tight bounds are known for the BKZ and DEEP algorithms. Furthermore, while exhaustive enumeration methods are not practical in higher ranks, they are used as a subroutine in BKZ to search for short vectors in blocks. Therefore, Gama and Nguyen examined the experimental running time of such methods as well.

For their experiments on exhaustive enumeration, Gama and Nguyen used a method due to Schnorr and Euchner [SE]. This method is used as a subroutine in BKZ and it seems to outperform other algorithms in practice, despite having worse theoretical bounds on its running time. On input of a lattice basis, the algorithm finds a shortest vector. The enumeration becomes faster as the input basis is more reduced. From their experiments, Gama and Nguyen note that SVP can be solved within an hour for rank $d = 60$, whereas the curve of their results shows that solving it for rank $d = 100$ would take at least 35000 years. This can be improved by better preprocessing such as basis reduction, but still Gama and Nguyen think that enumeration is not possible for lattices of rank $d \geq 100$.

The best known upper bound for the running time of BKZ is superexponential, while BKZ with blocksize $\beta = 20$ can reduce the basis of a lattice of rank $d = 100$ in a few seconds. This suggests that the superexponential bound is not tight. Gama and Nguyen measured the running time of BKZ in their experiments for several blocksizes and on lattices of varying rank. They observed that the running time increased exponentially with the blocksize, as expected. However, for blocksizes $20 \leq \beta \leq 25$, the running time started to increase disproportionately. The slope of the running time suddenly increased as seen on a logarithmic scale. This effect increased further for lattices of higher rank. It follows that BKZ with blocksize $\beta > 25$ is infeasible for lattices with high rank.

The running time of the DEEP algorithm seems to increase more regularly for increasing blocksize. It increases exponentially in the blocksize, just like the running time of BKZ. As opposed to BKZ, there is no

sharp increase for higher blocksizes. This allows for DEEP to be run on high-ranked lattices with relatively high blocksizes. However, the experimental results on the quality of the bases showed that even with higher blocksizes, the Hermite factor achieved by DEEP is not expected to improve significantly beyond 1.01^d.

6.2.4 Results for cryptography

What do these different results mean for cryptography? The experiments show that the Hermite and approximation factor that can be reached by basis reduction algorithms are exponential in the dimension, as was expected from the theory. However, the base of this exponential is much lower in practice than the theory predicts. This means that, although approximating these problems is still hard asymptotically, the lattice rank needs to be at least 500 before this hardness emerges. For instance, from the results of these experiments it follows that approximating either of these problems within a factor $\gamma = d$ is easy for $d \leq 450$, because the Hermite factor δ^d reached in practice is approximately linear in d. Furthermore, the experiments show that USVP is easy to solve whenever the gap λ_2/λ_1 is a fraction of the Hermite factor. However, by focusing on the base δ of the Hermite factor δ^d and considering what is feasible and what is not, some information is lost. The parameter δ^d predicts the quality of the resulting vectors in terms of the lattice rank d, but it does not say anything about the effort in relation to the rank of the lattice.

It should be noted that these results apply to the specific distribution of lattices described by Goldstein and Mayer [GM]. Unless the lattices that arise from cryptographic situations come from this distribution, some additional experiments are required to determine the actual performance of basis reduction algorithms on these 'cryptographic' lattices. Rückert and Schneider [RS] performed similar experiments for lattices that come from cryptographic examples. These results will be discussed in the next subsection.

Another point of interest is that these conclusions change as time goes on. As computing power increases, it becomes practical to break cryptosystems using lattices of higher rank. This should be taken into account when determining the performance of basis reduction algorithms, and especially in the context of lattice-based cryptography. Finally, improvements in the area of basis reduction will improve the performance of basis reduction algorithms. As with computing power, this affects the security of all lattice-based cryptosystems.

While Gama and Nguyen have given much insight into the practical behavior of basis reduction algorithms, there is still work to be done. The biggest downside to the method of Gama and Nguyen is that they only distinguish between lattice problems that are 'within reach' and those that are 'not within reach' given the current basis reduction algorithms and computing power. They do not provide a method to measure the actual cost or required effort of these algorithms, nor a way to predict how hard the problems will be in the future. In the next section, a framework that attempts to solve these problems for lattice-based cryptosystems will be discussed.

6.3 Security in practice

Inspired by the results of Gama and Nguyen, Rückert and Schneider analyzed the security of lattice-based cryptosystems using a similar approach. They consider lattices that come from cryptographic applications, aiming to create a framework to determine the practical security of cryptosystems based on the SIS and LWE problems from their parameters.

The idea of a unified framework to consider the security of all lattice-based cryptosystems is not entirely new. It was inspired by the works of Lenstra and Verheul [LV] and the subsequent update by Lenstra [Le]. The framework of Rückert and Schneider is explained in three steps. First, they show how to represent the hardness of the SIS and LWE problems with a single parameter. Then, they perform experiments to relate this parameter to the attack effort. Finally, they apply their framework to several cryptographic schemes to measure and compare their security.

Before the framework is explained, the notion of *attack effort* will be examined. Afterwards, the three steps

of the framework will be explained in more detail.

6.3.1 Measuring security

When measuring practical security, we need to take the capabilities of the attacker into account. Furthermore, advances in both computing power and cryptanalytic methods will increase these capabilities in the future. Therefore, Rückert and Schneider model the attacker in their framework as well. In order to measure the attack effort, they use the notion of dollar-days, which was introduced by Lenstra [Le]. Dollar-days are the cost of equipment in dollars multiplied by the time spent on the attack measured in days. Thus, using a \$1000 computer and spending 4 days to break a system costs as much in terms of dollar days as using a \$2000 computer and spending 2 days.

Consider a cryptographic system with security parameter k. Assume that the best known attack against this system requires $t(k)$ seconds on a computer that costs d dollars. The cost of this attack, as represented in dollar days, is given by

$$T(k) = d \cdot t(k)/(3600 \cdot 24).$$

If (an estimation of) the function $T(k)$ is known, recommended parameters can be chosen as follows. Assume an attacker has T_{y_0} dollar days at his disposal, where y_0 stands for a certain year. To be secure against this particular attacker, the security parameter of the system must exceed a value k^* such that $T(k^*) \geq T_{y_0}$.

It is also possible to consider future developments and estimate what the security of the system will be in the future. To this end, Rückert and Schneider consider a rule that Lenstra calls the "double Moore law". Moore's law states that the computing power doubles every 18 months. The double Moore law also takes advances in the field of cryptanalysis into account. Each year, the security is expected to degrade by a factor of $2^{-12/9}$. However, this function is based on algorithmic progress in the area of integer factorization. Rückert and Schneider adopt it because they find the algorithmic progress of lattice basis reduction hard to judge. To be secure up until year y against an attacker that has T_{y_0} dollar days at his disposal in year y_0, the security parameter must satisfy $T(k) \geq T_{y_0} \cdot 2^{(y-y_0)\cdot 12/9}$.

Some cryptographic schemes also use symmetric cryptographic primitives. Rückert and Schneider assume that these primitives are always available and that they are only affected by Moore's law. They reason that symmetric primitives can be replaced more easily when attacks are found than asymmetric ones.

6.3.2 Measuring the hardness of SIS and LWE

Both the SIS and LWE problems have several parameters that can influence their hardness. However, it is desirable to represent the security of cryptographic systems with a single security parameter. First, Rückert and Schneider analyze the hardness of the SIS problem, introducing a parameter that corresponds to the best known attack. Then, they provide some experimental data to show that this parameter influences the result the most. Finally, they show how to reduce LWE to SIS, which allows them to reuse the hardness results of SIS for LWE.

Recall from the description of SIS in Section 3 that it has the four parameters n, m, q and ν and that the corresponding lattice is of the form

$$\Lambda_q^{\perp}(A) = \{\mathbf{x} \in \mathbb{Z}^m : A\mathbf{x} = 0 \bmod q\},$$

where A is an $n \times m$ matrix with entries in $\mathbb{Z}_q$. If the rows of A are linearly independent, then $\Lambda_q^{\perp}(A)$ is a full-rank lattice in $\mathbb{Z}^m$. Furthermore, it contains q^{m-n} lattice vectors that are in $\mathbb{Z}_q^m$ and hence its volume is q^n.

Practical hardness is measured by the resistance against attacks. Thus, in order to measure the practical hardness of SIS, we must determine the best way to attack it. At this time, no methods are known to

perform better than basis reduction algorithms. Recall that basis reduction algorithms are δ-HSVP solvers that find a vector of norm at most $\delta^d \mathrm{vol}(L)^{1/d}$ in a lattice L, where d is the lattice rank.

The system $A\mathbf{x} = 0$ is underdetermined. Therefore, it is possible to fix k coordinates of $\mathbf{x}$ to zero, and attempt to solve the problem using the other $m-k$ coordinates. This results in a sublattice of $\Lambda_q^\perp(A)$ of lower rank, which still has volume q^n with high probability. Using this observation, Micciancio and Regev [MR2] introduced a sublattice attack on the SIS problem. Instead of trying to solve SIS in the lattice $\Lambda_q^\perp(A)$, they propose to solve it in the lattice $\Lambda_q^\perp(A')$, where A' is obtained by removing some columns from A. As the resulting lattice is contained in a space of lower dimension, the obtained vectors should be padded with zeroes where the columns of A were removed. Let A' have $m-k=d$ columns, which means the lattice $\Lambda_q^\perp(A')$ has rank d. Applying a basis reduction algorithm to the sublattice $\Lambda_q^\perp(A')$ gives a vector of norm at most $\delta^d \mathrm{vol}(\Lambda_q^\perp(A')) = \delta^d q^{n/d}$. Micciancio and Regev showed that the minimum of this function is obtained for $d = \sqrt{n \log_2 q / \log_2 \delta}$. For this d, δ satisfies the equation

$$\delta = 2^{n \log_2 q / d^2}$$

Using a sufficiently strong HSVP solver will result in a vector that has norm at most $\delta^d q^{n/d} = q^{2n/d}$.

Rückert and Schneider note that the above analysis does not include the parameter ν of the SIS problem. Micciancio and Regev assume that the δ of the HSVP solver is fixed, but an attacker might employ basis reduction algorithms of varying strength, until a suitably short vector is found. Therefore, they propose to take ν into account when determining the strongest attack. This results in the following theorem:

Theorem 6.1. *Let $n \geq 128$, $q \geq n^2$ and $\nu < q$. The optimal lattice rank for solving SIS(n,m,q,ν) with a δ-HSVP solver for variable δ is $d = \min\{x \in \mathbb{N} : q^{2n/x} \leq \nu\}$.*

In their proof, Rückert and Schneider show that the solver must be able to solve δ-HSVP for $\delta \leq \sqrt[d]{\nu / q^{n/d}}$. They also prove that the minimum attack rank d must be at least $2n = 256$, since if $d \leq 2n$ then $q \leq q^{2n/d} \leq \nu < q$, which gives a contradiction. They sum up the basis of their analysis in the following conjecture:

Conjecture 6.2. *Let $n > 128$, a constant $c \geq 2$, a prime $q \geq n^c$, $m = \Omega(n \log_2(q))$ and $\nu < q$ be given. Then, the best known approach to solve SIS(n,m,q,ν) is to solve δ-HSVP in rank $d = \min\{x \in \mathbb{N} : q^{2n/x} \leq \nu\}$ with $\delta = \sqrt[d]{\nu / q^{n/d}}$.*

This conjecture assumes that the best possible attack on the SIS-problem consists of solving the Hermite shortest vector problem in a suitable lattice. Now, Rückert and Schneider claim that the most natural approach to solve the decision version of the LWE-problem is by solving an instance of the SIS problem. By reducing LWE to SIS, they can use hardness estimates for SIS to provide hardness estimates for LWE. This reduction was mentioned in Section 3 and is formalized in the following theorem:

Theorem 6.3. *Any algorithm that solves SIS with the parameters $(n,q,m,\nu = 1.5\sqrt{2\pi}/\alpha)$ can be used to solve LWE with parameters (n,q,m,α).*

Next, Rückert and Schneider performed experiments to see the influence of the different parameters on the hardness of the SIS problem.

6.3.3 Experimental data

In the experiments, Rückert and Schneider apply BKZ (as implemented in NTL [Sho]) to sublattices of the optimal rank d (as defined by Theorem 6.1). They gradually increase the blocksize β of BKZ – thus decreasing δ – until a vector of the desired length is found. Then, they measure the running time of the reduction algorithm and compare these for varying n, m and q.

Their first observation is that for $\delta \in (1, 1.02]$, q has but a relatively minor impact on the running time of the reduction algorithm. Secondly, they note that the lattice rank m influences the running time more

noticeably. However, they claim that the most influential parameter is δ. For $\delta < 1.015$, the running time increases rapidly as δ decreases. Thus, they conclude that δ should be considered the main security parameter.

Finally, in order to obtain the security estimates, Rückert and Schneider fix $m = 175$, $n > 128$ and $q \approx n^3$. With δ as the main security parameter, they consider the cost function to be $T(\delta) = a2^{1/(\log_2(\delta)^b)} + c$ dollar days, where a, b and c are constants. Next, they use the experimental data to approximate the constants a, b and c, resulting in the values $a \approx 10^{-15}$, $b \approx 1.001$ and $c = 0.005$. They consider c to be negligible for small δ, which leads to the following conjecture:

Conjecture 6.4. *Let $n > 128$, a constant $c \geq 2$, a prime $q \geq n^c$, $m = \Omega(n\log_2(q))$ and $\nu < q$ be given. Then, for any $\delta \in (1, 1.015]$, solving δ-HSVP in (normalized) q-ary lattices of rank d costs at least $T(\delta) = 10^{-15}2^{1/(\log_2(\delta)^{1.001})}$ dollar days.*

Using this cost function, Rückert and Schneider predict parameters δ for HSVP that are infeasible for certain attackers. They distinguish between the three classes of attackers, inspired by works of Blaze et al. [BDRSSTW] and Lenstra [Le]. These classes are Hacker, Lenstra and Intelligence agency, each having a different budget. The Hacker has access to 400 dollar days, the attacker Lenstra possesses 40 million dollar days and the Intelligence agency has 108 billion dollar days at its disposal. The infeasible values for δ that they computed using the cost function are shown in Table 2. The table also includes values for the corresponding bit security. The derivation of the bit security follows from the work of Lenstra and Verheul. It is computed by the formula $\lceil 56 + 12(y - 1982)/18 \rceil$, where y is the year. The significance of this formula is that 56 is the bit security of DES, which was considered to be secure until the year 1982, "even against the strongest attacker". The factor $12/18$ follows from the simple Moore law, and thus the formula is based on the assumption that DES was secure in 1982 and that since then, attackers have become able to break at most $12(y - 1982)/18$ more bits of security.

Year	2010	2020	2030	2040	2050	2060	2070	2080	2090	2100
Bit security	75	82	88	95	102	108	115	122	128	135
Hacker	1.01177	1.00965	1.00808	1.00702	1.00621	1.00552	1.00501	1.00458	1.00419	1.00389
Lenstra	1.00919	1.00785	1.00678	1.00602	1.00541	1.00488	1.00447	1.00413	1.00381	1.00356
Int. Agency	1.00799	1.00695	1.00610	1.00548	1.00497	1.00452	1.00417	1.00387	1.00359	1.00336

Table 2: Values of δ predicted to be infeasible to break for the attackers.

Sadly, this does not give a direct *relation* between bit security and the feasibility of lattice problems. It merely lists values of δ such that breaking some cryptosystem is infeasible for a given attacker in a given year next to the number of bits such that breaking a symmetric algorithm with this key length is infeasible for all attackers in a given year. It would be interesting to consider a more direct relation between the effort required to achieve such δ and the effort required to break a system with such a key length.

Furthermore, the decision to consider only δ for the effort function T, while ignoring parameters such as the dimension n and the lattice rank m seems questionable. Even if the effect of δ on the effort is much more noticeable than the effect of other parameters such as m and n, it is still interesting to consider the effects of the parameters δ, m and n on the efficiency of the cryptosystem. It might be that changing the δ is much more costly than changing the m or n parameters. In any case, it seems prudent to keep the effort function more general, and include other parameters as well. However, it should be noted that this will increase the complexity of the model.

6.4 Learning With Errors in practice

In order to get a single framework for all cryptosystems based on the SIS and LWE problems, Rückert and Schneider assume that the best attack on cryptosystems based on the LWE problem is the so-called distinguishing attack, which uses an SIS-solver to solve the decision-LWE problem. Lindner and Peikert [LP] show that this distinguishing attack on the decision-LWE problem may be more costly than an attack on

the search-LWE problem, which is arguably a 'harder' problem. They propose and analyze such an attack and then compare it to the distinguishing attack. We will describe and discuss their results here.

6.4.1 Distinguishing versus decoding

In the distinguishing attack, the attacker tries to solve the SIS problem in the scaled dual lattice $\Lambda^{\perp}(A)$ of the LWE lattice $\Lambda(A)$. Then, he uses the short vector that he found to distinguish the LWE lattice $\Lambda(A)$ from being uniformly random. Specifically, he tries to find a short vector $\mathbf{v}$ such that

$$A\mathbf{v} = 0 \bmod q.$$

Now, given an LWE instance $A^t\mathbf{s} + \mathbf{e} = \mathbf{t}$, the attacker can compute

$$\langle \mathbf{v}, \mathbf{t} \rangle = \mathbf{v}^t A^t \mathbf{s} + \langle \mathbf{v}, \mathbf{t} \rangle = \langle \mathbf{v}, \mathbf{t} \rangle \bmod q,$$

which roughly behaves as a Gaussian $\bmod q$ with parameter $\|\mathbf{v}\| \cdot s$ if $\mathbf{e}$ is Gaussian with error parameter s. If $\mathbf{v}$ is short enough, this Gaussian can be distinguished from uniformly random with an advantage of approximately $\exp(-\pi \cdot (\|\mathbf{v}\| \cdot s/q)^2)$. However, Lindner and Peikert observed that this attack might be more costly than an attack that retrieves the LWE secret $\mathbf{s}$, especially for high advantages. The reason is that the vector $\mathbf{v}$ needs to be quite short and basis reduction algorithms can get very costly for such approximation factors. The attack that Lindner and Peikert propose works with bases of lesser quality as well.

This attack consists of two steps: a basis reduction step and a decoding step. The decoding step tries to solve a Closest Vector Problem (or rather a Bounded Distance Decoding problem) and its effectiveness depends on the quality of the basis. To analyze the first step, Lindner and Peikert perform experiments with BKZ on q-ary lattices, in an attempt to confirm the results of Gama and Nguyen for these kinds of lattices. They also check if the resulting bases adhere to the *Geometric Series Assumption* (GSA), which says that after performing basis reduction, the norms of the Gram-Schmidt vectors will form a geometrically decreasing sequence. This assumption was introduced by Schnorr [Sno2].

The decoding step consists of a variant of Babai's Nearest Plane algorithm and can be seen as a form of lattice vector enumeration. Recall how Babai's Nearest Plane algorithm works from Section 2. Given a target vector $\mathbf{t}$ and a lattice basis $\mathbf{b}_1, \ldots, \mathbf{b}_m$ it finds a lattice vector $\mathbf{u} = \sum_{i=1}^{m} u_i \mathbf{b}_i$ relatively close to $\mathbf{t}$ by greedily choosing the coefficients $u_m, u_{m-1}, \ldots, u_1$ in that order. For all k, Babai's algorithm chooses the coefficient $u_k = \left\lceil t_k - \sum_{i=k+1}^{m} \mu_{i,k}(u_i - t_i) \right\rfloor$. Let $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$ be the Gram-Schmidt orthogonalisation of the basis. Now consider the following representation of the difference vector $\mathbf{u} - \mathbf{t}$

$$\begin{aligned} \mathbf{u} - \mathbf{t} &= \sum_{i=1}^{m} (u_i - t_i)\mathbf{b}_i \\ &= \sum_{j=1}^{m} \left(u_j - t_j + \sum_{i=j+1}^{m} (u_i - t_i)\mu_{i,j} \right) \mathbf{b}_j^*. \end{aligned}$$

Inserting the choice of u_j in the previous equation shows that the coefficient of $\mathbf{b}_j^*$ in $\mathbf{u} - \mathbf{t}$ is equal to $\left(\left\lceil t_j - \sum_{i=j+1}^{m} \mu_{i,j}(u_i - t_i) \right\rfloor - t_j + \sum_{i=j+1}^{m} (u_i - t_i)\mu_{i,j} \right)$ which is at most 1/2 in absolute value. Thus, Babai's Nearest Plane algorithm outputs the unique lattice vector that lies in the parallelepiped spanned by the Gram-Schmidt vectors around $\mathbf{t}$.

Lindner and Peikert note that the GSA implies that after performing basis reduction, the last Gram-Schmidt vectors will be relatively small compared to the first ones. Thus, this parallelepiped that appears in Babai's Nearest Plane algorithm will be very thin in the direction of the last Gram-Schmidt vectors and very long in the direction of the first ones. Since the error vector in the LWE problem is a discrete Gaussian in most cases, it is equally likely to go in either direction. Thus, it makes sense to extend the search area in the direction of the smaller Gram-Schmidt vectors.

The algorithm that Lindner and Peikert propose considers extra branchings in the enumeration tree. At each level in the tree, they choose a parameter d_i equal to the number of branches (corresponding to the coefficient of $\mathbf{b}_i$) they will search at this level. This means that for each node at level $d-i$ in the tree, the algorithm searches d_i children at level $d-i+1$. The result is that the algorithm goes through all vectors in the area

$$\mathbf{t} + \mathscr{P}_{1/2}(D \cdot B^*),$$

where D is the matrix that contains d_i as its diagonal elements. This algorithm increases the running time of Babai's Nearest Plane algorithm by a factor of $\prod_i d_i$. Lindner and Peikert also compute the probability that the algorithm finds the closest lattice vector:

$$\mathbb{P}(\mathbf{e} \in \mathscr{P}_{1/2}(D \cdot B^*) = \prod_{i=1}^{m} \operatorname{erf}\left(\frac{d_i \|\mathbf{b}_i^*\| \sqrt{\pi}}{2s}\right). \tag{6.1}$$

This equation suggests that an attacker should choose his d_i to maximize the minimum of $d_i\|\mathbf{b}_i^*\|$, while keeping the total cost $\prod_i d_i$ low.

6.4.2 Analyzing BKZ

In order to analyze BKZ, Lindner and Peikert perform experiments on modular lattices arising from the LWE problem and its associated dual SIS problem. From these experiments, they observe the same behavior as Gama and Nguyen: the Hermite factor is the dominant parameter in both the runtime and quality of the basis when using BKZ. Therefore, they extrapolate the runtime of BKZ as a function of δ, using as a rule of thumb that obtaining an approximation within a factor 2^k of the shortest vector in an m-dimensional lattice takes time $2^{\tilde{O}(m/k)}$ using BKZ. This suggests that the logarithmic running time t_{BKZ} of BKZ should grow roughly linearly in $1/\log_2(\delta)$. They use least-square regression to fit their data to a linear function, and this results in a logarithmic cost of $t_{\text{BKZ}} = 1.806/\log_2(\delta) - 91$. However, they concede that their experiments were limited by resources and available time and hence use the conservative lower bound estimate of

$$t_{\text{BKZ}} = 1.8/\log_2(\delta) - 110.$$

This relation allows them to choose a δ which fixes the root-Hermite factor of the shortest vector found by BKZ, as well as the running time of BKZ.

However, for the second step of the attack, the attacker uses the whole basis, rather than just the shortest basis vector. How can we say something about the shape of the whole basis from δ, the root-Hermite factor of the shortest basis vector? This is where the Geometric Series Assumption comes in. It says that after performing BKZ, the lengths $\|\mathbf{b}_i^*\|$ of the Gram-Schmidt vectors decay geometrically with i, i.e., $\|\mathbf{b}_{i+1}^*\| = \alpha \cdot \|\mathbf{b}_i^*\|$ for some $0 < \alpha < 1$. As a result, all Gram-Schmidt lengths can be computed from the length of the first vector, since $\|\mathbf{b}_i^*\| = \alpha^{i-1} \cdot \|\mathbf{b}_1\|$.

6.4.3 Attack

Combining these observations, Lindner and Peikert analyze the attack as follows: for a given advantage, they pick a δ, which gives them the optimal dimension m, the running time of BKZ as well as the shape of the reduced basis (in terms of the lengths of Gram-Schmidt vectors). Then, they pick the d_i's of the second step such that the desired advantage is achieved by computing the probability of success using (6.1). They only consider the attack successful if the cost of this second step, given by the product of the d_i's, does not exceed that of BKZ. Now, they do this for several values of δ and pick the one that leads to the lowest overall running time for the given advantage.

Parameters				Distinguish		Decode		
n	q	s	$\log_2(1/\varepsilon)$	δ	$\log_2$(secs)	δ	$\log_2$(# enum)	$\log_2$(secs
			≈ 0	*1.0065	83	1.0089	47	32
128	2053	6.77	-32	1.0115	< 0	1.0116	13	< 0
	(toy)		-64	1.0128	< 0	1.0130	1	< 0
			≈ 0	*1.0045	168	1.0067	87	78
192	4093	8.87	-32	1.0079	49	1.0083	54	42
	(low)		-64	1.0097	34	1.0091	44	29
			≈ 0	*1.0034	258	*1.0058	131	132
256	4093	8.35	-32	1.0061	96	1.0063	87	90
	(medium)		-64	1.0067	77	1.0068	73	75
			≈ 0	*1.0027	353	*1.0042	163	189
320	4093	8.00	-32	1.0049	146	1.0052	138	132
	(high)		-64	1.0054	122	1.0055	117	119

Table 3: Comparison of the distinguishing and decoding attacks.

Their results are compiled in Table 3, where they compare the cost of this attack to that of the distinguishing attack for several advantages and parameter choices.

The main conclusion from these results is that the distinguishing attack may not always be better than solving the search problem. For small advantages, the required values of δ appear to be very close, whereas for high advantages the distinguishing attack requires lower values of δ. A lower δ does not only increase the cost of BKZ substantially, it also increases the optimal dimension m for the attack. In the same paper, Lindner and Peikert introduce a system which only releases a limited number of LWE samples, which means that in some cases, this optimal dimension is not attainable due to a lack of LWE samples. Note that if we want to verify these results by practically performing these attacks, we would still need to know which blocksize (and possibly other parameters) we should choose for BKZ to achieve the given δ.

6.5 BKZ 2.0

In Section 4.4 we discussed the BKZ algorithm. Recently, some improvements have been proposed by Chen and Nguyen [CN]. The first improvement consists of incorporating the extreme pruning algorithm into BKZ, which provides an exponential speedup to the enumeration. This improvement allows them to perform experiments with BKZ using higher blocksizes than deemed possible from the experiments of Gama and Nguyen.

Because of the higher blocksizes, the enumerations were performed in projected lattices of higher dimensions. It turns out that, unlike for lower blocksizes, these projected lattices behave like random lattices in the sense that the Gaussian Heuristic holds. Thus, the length of a shortest vector is roughly what we expect from the Gaussian Heuristic. The second improvement consists of choosing the expected length of the shortest vector from the Gaussian Heuristic as the enumeration bound, as opposed to the length of the first projected basis vector in the block. Of course, this is only done if the expected length is smaller than the length of the current first block vector.

Finally, Chen and Nguyen observe that during BKZ, the following happens. BKZ is gradually improving the 'quality' of the overall basis by finding shorter vectors. At some point, the enumeration routine finds a short vector and inserts it into the basis. Now, LLL is performed to remove the linear dependency created by the extra vector. Thus, the enumeration of the next block will be performed on a basis that is locally only LLL reduced, while the basis of other blocks is better reduced. Thus, they propose to reduce the basis of the block better before each enumeration. This can significantly increase the enumeration time for higher blocksizes.

These improvements increase the quality of the output basis (again measured in terms of the root-Hermite

factor δ), but more importantly, the observation about the random behavior of the projected lattices allowed Chen and Nguyen to analyze BKZ from a theoretical point of view rather than a purely experimental one. The fact that the shortest vectors in every block are roughly of the expected length prompted them to simulate BKZ in an iterative manner. We shall describe the general ideas behind this simulation here. The simulation takes a blocksize β, a number of rounds N and the log Gram-Schmidt lengths $\log(\|\mathbf{b}_i^*\|)$ as input and iteratively performs a single BKZ round on the input basis until it has performed N rounds. In each round, the effect of every single enumeration is simulated as follows: Compute the volume of the enumeration block and use the Gaussian Heuristic to estimate the length of the shortest vector. If this length is smaller than the length of the current first block vector, the length of the current first block vector is replaced by the new shorter length. For the last β vectors the simulation does something extra, because these vectors will be part of an HKZ-reduced basis at the end of the round. The last 45 lengths are set to the lengths of a random HKZ-reduced basis of dimension 45 with the appropriate volume and the other $\beta - 45$ lengths are set according to the Gaussian Heuristic of lattices with the appropriate volume and dimension.

The simulation outputs the log Gram-Schmidt lengths $\log(\|\mathbf{b}_i^*\|)$ of the resulting basis, which can be used to compute the root-Hermite factor δ. Thus, this simulation provides us with a way to relate blocksize and number of rounds to reduction quality. Surprisingly, Chen and Nguyen note that "for a given dimension n, one should run the simulation algorithm, because the actual blocksize also depends on the dimension", but they do not elaborate on the nature of this dependency.

They also have some results on the runtime of BKZ. If the algorithm is aborted after a fixed number of rounds N, we know exactly how many enumerations will be performed. Thus, it remains to quantify how expensive these enumerations are. Chen and Nguyen give both upper and lower bounds on the number of nodes in enumeration trees. For their upper bounds, they apply extreme pruning on bases reduced with BKZ 2.0.

Blocksize	100	110	120	130	140	150	160	170	180	190	200	250
BKZ-75-20%	41.4	47.1	53.1	59.8	66.8	75.2	84.7	94.7	105.8	117.6	129.4	204.1
BKZ-90/100/110/120	40.8	45.3	50.3	56.3	63.3	69.4	79.9	89.1	99.1	103.3	111.1	175.2

Table 4: Upper bounds on the log number of nodes in the enumeration tree for varying blocksizes.

Table 4 contains the upper bounds, in log number of nodes. The first row gives upper bounds on the log number of nodes in the enumeration tree of bases that are reduced with aborted BKZ with blocksize 75 and a pruning parameter of 20%. The second row also gives upper bounds, but here the simulation is used to simulate the performance of BKZ with blocksize 90 up to 120 on the blocks before the enumeration. For the lower bounds, they use some earlier work by Nguyen that gives a lower bound on the number of nodes in the middle depth (at level $\beta/2$) of a full enumeration tree. To get similar results for pruned trees, they merely divide through by the asymptotic speedups of these pruning techniques ($2^{n/4}$ for linear pruning and $2^{n/2}$ for extreme pruning). The resulting bounds are given in Table 5.

Blocksize	100	120	140	160	180	200	220	240	280	380
Linear pruning	33.6	44.5	56.1	68.2	80.7	93.7	107.0	120.6	148.8	223.5
Extreme pruning	9	15	21.7	28.8	36.4	44.4	52.8	61.5	79.8	129.9

Table 5: Lower bound on the log number of nodes in the enumeration tree for varying blocksizes.

Acknowledgements

This paper grew out of two presentations given at the VI-MSS workshop "Mathematical and Statistical Aspects of Cryptography", held January 12–14, 2012 at the Indian Statistical Institute, Kolkata, India.

The first author was sponsored by DIAMANT/NWO and ECRYPT II and VI-MSS.
The second author was sponsored in part by EPSRC via grant EP/I03126X and by VI-MSS.
The third author was sponsored by ECRYPT II and VI-MSS.

References

[AD] M. Ajtai and C. Dwork, *A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence*, Proc. 29th Symp. Theory of Computing (STOC), pp. 284–293, 1997.

[AKS] M. Ajtai, R. Kumar, and D. Sivakumar, *A Sieve Algorithm for the Shortest Vector Problem*, Proc. 33rd Symp. Theory of Computing (STOC), pp. 601–610, 2001.

[ABSS] S. Arora, L. Babai, J. Stern and Z. Sweedyk, *The hardness of approximate optima in lattices, codes, and systems of linear equations*, in: Proc. of the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993, pp. 724–733.

[B] L. Babai, *On Lovász Lattice Reduction and the Nearest Lattice Point Problem*, Combinatorica, vol. 6, no. 1, pp. 1–13, 1986.

[BBD] D.J. Bernstein, J. Buchmann and E. Dahmen (Eds.), Post-Quantum Cryptography, Springer, 2009.

[BDRSSTW] M. Blaze, W. Diffie, R.L. Rivest, B. Schneier, T. Shimomura, E. Thompson and M. Wiener, *Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security. A Report by an Ad Hoc Group of Cryptographers and Computer Scientists*, pp. 1–10, 1996.

[BS] J. Blömer and J.P. Seifert, *On the complexity of computing short linearly independent vectors and short bases in a lattice*, in Proc. of the 31st annual ACM symposium on Theory of Computing, pp. 711–720, ACM, 1999.

[CN] Y. Chen and P.Q. Nguyen, *BKZ 2.0: Better Lattice Security Estimates*, Proc. 17th Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT), pp. 1–20, 2011.

[EB] , P. van Emde Boas, *Another NP-complete partition problem and the complexity of computing short vectors in a lattice*, Technical Report 81-04, Mathematisch Instituut, Universiteit van Amsterdam, 1981.

[E] Euclid, *Elements*, ca. 300 BC.

[FP] U. Fincke and M. Pohst, *Improved Methods for Calculating Vectors of Short Length in a Lattice*, Mathematics of Computation, vol. 44, no. 170, pp. 463–471, 1985.

[GN] N. Gama and P.Q. Nguyen, *Predicting Lattice Reduction*, Proc. 27th Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT), pp. 31–51, 2008.

[GNR] N. Gama, P.Q. Nguyen and O. Regev, *Lattice Enumeration using Extreme Pruning*, Proc. 29th Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT), pp. 257–278, 2010.

[Ga] C.F. Gauss, *Disquisitiones Arithmeticae*, Leipzig, 1801.

[Ge] C. Gentry, *Fully Homomorphic Encryption using Ideal Lattices*, Proc. 41st Symp. Theory of Computing (STOC), pp. 169–178, 2009.

[GGH1] O. Goldreich, S. Goldwasser and S. Halevi, *Collision-Free Hashing from Lattice Problems*, Electronic Colloquium on Computational Complexity, vol. 3, no. 42, pp. 1–10, 1996.

[GGH2] O. Goldreich, S. Goldwasser and S. Halevi, *Public-key Cryptosystems from Lattice Reduction Problems*, Proc. 17th Cryptology Conference (CRYPTO), pp. 112–131, 1997.

[GMSS] O. Goldreich, D. Micciancio, S. Safra and J.P. Seifert, *Approximating Shortest Lattice Vectors is Not Harder Than Approximating Closest Lattice Vectors*, Information Processing Letters, vol. 71, no. 2, pp. 55–61, 1999.

[GM] D. Goldstein and A. Mayer, *On the equidistribution of Hecke points*, Forum Mathematicum, vol. 15, no. 2, pp. 165–189, 2003.

[HPS1] G. Hanrot, X. Pujol and D. Stehlé, *Algorithms for the Shortest and Closest Vector Problems*, Proc. 3rd Int. Workshop on Coding and Cryptology (IWCC), pp. 159–190, 2011.

[HPS2] G. Hanrot, X. Pujol and D. Stehlé, *Analyzing Blockwise Lattice Algorithms using Dynamical Systems*, Proc. 31th Cryptology Conference (CRYPTO), pp. 447–464, 2011.

[HS1] G. Hanrot and D. Stehlé, *Improved Analysis of Kannan's Shortest Lattice Vector Algorithm*, Proc. 27th Cryptology Conference (CRYPTO), pp. 170–186, 2007.

[HS2] G. Hanrot and D. Stehlé, *A Complete Worst-Case Analysis of Kannan's Shortest Lattice Vector Algorithm*, Submitted, pp. 1–34, 2010.

[H] C. Hermite, *Extraits de lettres de M. Hermite à M. Jacobi sur diff'erent objets de la théorie des nombres, deuxième lettre*, J. Reine Angew. Math. **40** [1850], 279–290.

[HPS] J. Hoffstein, J. Pipher and J.H. Silverman, *NTRU: A Ring-Based Public Key Cryptosystem*, Proc. 3rd Symp. Algorithmic Number Theory (ANTS), pp. 267–288, 1998.

[KL] G.A. Kabatiansky and V.I. Levenshtein, *On Bounds for Packings on a Sphere and in Space (Russian)*, Problemy Peredachi Informacii, vol. 14, no. 1, pp. 3–25, 1978.

[Ka] R. Kannan, *Improved Algorithms for Integer Programming and Related Lattice Problems*, Proc. 15th Symp. Theory of Computing (STOC), pp. 193–206, 1983.

[Kh] S. Khot, *Hardness of approximating the shortest vector problem in lattices*, Journal of the ACM **52** [2005], 789–808.

[KZ] A. Korkine and G. Zolotarev, *Sur les formes quadratiques*, Math. Ann., vol. 6, no. 3, pp. 366–389, 1873.

[LO] J.C. Lagarias and A.M. Odlyzko, *Solving low-density subset sum problems*, J. ACM, vol. 32, no. 1, pp. 229–246, 1985.

[La] J.L. Lagrange, *Recherches d'Arithmétique*, Nouv. Mém. Acad., 1773.

[Le] A.K. Lenstra, *Key Lengths*, in The Handbook of Information Security, ch. 114, 2005.

[LLL] A.K. Lenstra, H.W. Lenstra and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Annalen, vol. 261, no. 4, pp. 515–534, 1982.

[LV] A.K. Lenstra and E.R. Verheul, *Selecting Cryptographic Key Sizes*, Journal of Cryptology, vol. 14, no. 4, pp. 255–293, 2001.

[LP] R. Lindner and C. Peikert, *Better Key Sizes (and Attacks) for LWE-based Encryption*, Topics in Cryptology (CT-RSA), pp. 319–339, 2011.

[LLLS] C. Ling, S. Liu, L. Luzzi and D. Stehlé, *Decoding by embedding: Correct decoding radius and DMT optimality*, IEEE International Symposium on Information Theory Proceedings (ISIT), pp. 1106–1110, 2011.

[LLM] Y.K. Liu, V. Lyubashevsky, and D. Micciancio, *On bounded distance decoding for general lattices*, Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, pp. 450–461, 2006.

[Lo] L. Lovász, *An algorithmic theory of numbers, graphs and convexity*, CBMS-NSF Reg. Conf. Ser. Appl. Math., vol. 50, pp. 91, 1986.

[LM] V. Lyubashevsky, D. Micciancio, *On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem*, Proc. 29th Cryptology Conference (CRYPTO), pp. 577–594, 2009.

[LPR] V. Lyubashevsky, C. Peikert and O. Regev, *On Ideal Lattices and Learning with Errors over Rings*, Proc. 29th Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT), pp. 1–23, 2010.

[Mic] D. Micciancio, *Efficient Reductions Among Lattice Problems*, Proc. 19th Symp. on Discrete Algorithms (SODA), pp. 84–93, 2008.

[MG] D. Micciancio and S. Goldwasser, *Complexity of Lattice Problems: A Cryptographic Perspective*, Kluwer, 2002.

[MR1] D. Micciancio and O. Regev, *Worst-Case to Average-Case Reductions Based on Gaussian Measures*, SIAM J. Comput., vol. 37, no. 1, pp. 267–302, 2007.

[MR2] D. Micciancio and O. Regev, *Lattice-Based Cryptography*, in [BBD], pp. 147–191, 2009.

[MV1] D. Micciancio and P. Voulgaris, *A Deterministic Single Exponential Time Algorithm for Most Lattice Problems based on Voronoi Cell Computations*, Proc. 42nd Symp. Theory of Computing (STOC), pp. 351–358, 2010.

[MV2] D. Micciancio and P. Voulgaris, *Faster Exponential Time Algorithms for the Shortest Vector Problem*, Proc. 21st Symp. on Discrete Algorithms (SODA), pp. 1468–1480, 2010.

[Min] H. Minkowski, *Geometrie der Zahlen*, Teubner, Leipzig, 1896.

[N] P.Q. Nguyen, *Hermites Constant and Lattice Algorithms*, in [NVa], pp. 19–69, 2009.

[NS1] P.Q. Nguyen and D. Stehlé, *Low-Dimensional Lattice Basis Reduction Revisited*, Proc. 6th Symp. Algorithmic Number Theory (ANTS), pp. 338–357, 2004.

[NS2] P.Q. Nguyen and D. Stehlé, *Floating-Point LLL Revisited*, Proc. 24th Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT), pp. 215–233, 2005.

[NS3] P.Q. Nguyen and D. Stehlé, *LLL on the Average*, Proc. 7th Symp. Algorithmic Number Theory (ANTS), pp. 238–256, 2006.

[NS4] P.Q. Nguyen and D. Stehlé, *An LLL Algorithm with Quadratic Complexity*, SIAM J. Comput., vol. 39, no. 3, pp. 874–903, 2009.

[NVa] P.Q. Nguyen and B. Vallée (Eds.), *The LLL Algorithm: Survey and Applications*, Springer-Verlag, 2009.

[NVi] P.Q. Nguyen and T. Vidick, *Sieve Algorithms for the Shortest Vector Problem are Practical*, J. Math. Cryptology, vol. 2, no. 2, pp. 181–207, 2008.

[Poh] M. Pohst, *On the Computation of Lattice Vectors of Minimal Length, Successive Minima and Reduced Bases with Applications*, ACM SIGSAM Bulletin, vol. 15, no. 1, pp. 37–44, 1981.

[Pol] J. van de Pol, *Lattice-based cryptography*, Master Thesis, Eindhoven University of Technology, 2011.

[PS1] X. Pujol and D. Stehlé, *Rigorous and Efficient Short Lattice Vectors Enumeration*, Proc. 14th Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT), pp. 390–405, 2008.

[PS2] X. Pujol and D. Stehlé, *Solving the Shortest Lattice Vector Problem in Time* $2^{2.465n}$, Cryptology ePrint Archive, Report 2009/605, pp. 1–7, 2009.

[R1] O. Regev, *On Lattices, Learning with Errors, Random Linear Codes, and Cryptography*, Proc. 37th Symp. Theory of Computing (STOC), pp. 84–93, 2005.

[R2] O. Regev, *Lattice-Based Cryptography*, Proc. 26th Cryptology Conference (CRYPTO), pp. 131–141, 2006.

[R3] O. Regev, *The Learning with Errors Problem (Invited Survey)*, Proc. 25th Conf. Computational Complexity (CCC), pp. 191–204, 2010.

[RS] M. Rückert and M. Schneider, *Estimating the Security of Lattice-based Cryptosystems*, Cryptology ePrint Archive, Report 2010/137, pp. 1–33, 2010.

[Sne1] M. Schneider, *Analysis of Gauss-Sieve for Solving the Shortest Vector Problem in Lattices*, Proc. 5th Workshop on Algorithms and Computation (WALCOM), pp. 89–97, 2011.

[Sne2] M. Schneider, *Sieving for Short Vectors in Ideal Lattices*, Cryptology ePrint Archive, Report 2011/458, pp. 1–19, 2011.

[Sno1] C.P. Schnorr, *A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms*, Theoretical Computer Science, vol. 53, no. 2–3, pp. 201–224, 1987.

[Sno2] C.P. Schnorr, *Lattice reduction by random sampling and birthday methods*, Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 145–56, 2003.

[Sno3] C.P. Schnorr, *Progress on LLL and Lattice Reduction*, in [NVa], pp. 145–178, 2009.

[Sno4] C.P. Schnorr, *Average Time Fast SVP and CVP Algorithms for Low Density Lattices and the Factorization of Integers*, Technical Report, pp. 1–18, 2010.

[SE] C.P. Schnorr and M. Euchner, *Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems*, Mathematical Programming, vol. 66, no. 2–3, pp. 181–199, 1994.

[SH] C.P. Schnorr and H.H. Hörner, *Attacking the Chor-Rivest Cryptosystem by Improved Lattice Reduction*, Proc. 14th Conf. Theory and Applications of Cryptographic Techniques (EUROCRYPT), pp. 1–12, 1995.

[Sha] C.E. Shannon, *Probability of Error for Optimal Codes in a Gaussian Channel*, Bell System Technical Journal, vol. 38, no. 3, pp. 611–656, 1959.

[Sho] V. Shoup, *Number Theory C++ Library (NTL)*, Available at `http://www.shoup.net/ntl/`, 2012.

[St] D. Stehlé, *Floating-Point LLL: Theoretical and Practical Aspects*, in [NVa], pp. 179–213, 2009.

[SSTX] D. Stehlé, R. Steinfeld, K. Tanaka and K. Xagawa, *Efficient Public Key Encryption Based on Ideal Lattices*, Proc. 15th Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT), pp. 617–635, 2009.

[VV] B. Vallée and A. Vera, *Probabilistic Analyses of Lattice Reduction Algorithms*, in [NVa], pp. 71–143, 2009.

[WLTB] X. Wang, M. Liu, C. Tian and J. Bi, *Improved Nguyen-Vidick Heuristic Sieve Algorithm for Shortest Vector Problem*, Proc. 6th Symp. Information, Computer and Communications Security (ASIACCS), pp. 1–9, 2011.