On the concrete hardness of Learning with Errors

Martin R. Albrecht[1], Rachel Player[1], and Sam Scott[1]

Information Security Group, Royal Holloway, University of London

Abstract. The Learning with Errors (LWE) problem has become a central building block of modern cryptographic constructions. This work collects and presents hardness results for concrete instances of LWE. In particular, we discuss algorithms proposed in the literature and give the expected resources required to run them. We consider both generic instances of LWE as well as small secret variants. Since for several methods of solving LWE we require a lattice reduction step, we also review lattice reduction algorithms and propose a refined model for estimating their running times. We also give concrete estimates for various families of LWE instances, provide a Sage module for computing these estimates and highlight gaps in the knowledge about algorithms for solving the Learning with Errors problem.

1 Introduction

Lattice-based cryptography. Lattice-based cryptography has become popular in recent years for several reasons. One dates back to the work of Ajtai [Ajt96] who linked the average case complexity of lattice problems to the worst case, showing that a random instance of the shortest vector problem in a lattice is hard. A second reason is its potential application in a post-quantum world, since no efficient quantum algorithms are known for lattice problems. On the other hand, problems such as factoring and discrete logarithm would no longer be hard in the presence of a quantum computer [BBD09]. A third reason is the wealth of applications of lattice-based cryptography, perhaps the most notably of which is its role in the realisation of fully homomorphic encryption by Gentry [Gen09] and in follow up works (e.g. [vDGHV10,BGV12,GHS12a]). Lattice problems have also been the basis of Public Key Encryption [Reg05], including CCA secure schemes [Pei09]; Identity Based Encryption [GPV08] and the more general Hierarchical Identity Based Encryption [CHKP12]; oblivious transfer schemes [PVW08]; Circular-Secure Encryption [ACPS09]; and Leakage-Resilient Encryption [GKPV10]. Recently, candidate constructions for multi-linear maps based on presumed hard problems in lattices have been proposed [GGH13,GGH14].

Learning with Errors. One lattice problem on whose hardness several cryptosystems are based is the Learning with Errors (LWE) problem [Reg05,Pei09,PW11]. LWE was introduced by Regev in [Reg05] and is provably as hard as worst-case lattice problems [Reg05,BLP+13]. It is a generalisation of the Learning Parity with Noise (LPN) problem into larger moduli q.

Definition 1 (LWE [Reg09]). *Let n, q be positive integers, χ be a probability distribution on $\mathbb{Z}$ and $\mathbf{s}$ be a secret vector in $\mathbb{Z}_q^n$. We denote by $L_{\mathbf{s},\chi}$ the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \in \mathbb{Z}$ according to χ, and returning $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.*
Decision-LWE *is the problem of deciding whether pairs $(\mathbf{a}, c) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ are sampled according to $L_{\mathbf{s},\chi}$ or the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.*
Search-LWE *is the problem of recovering $\mathbf{s}$ from $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ sampled according to $L_{\mathbf{s},\chi}$.*

Contributions. The first contribution of this survey is to gather and present algorithms available in the literature used for solving LWE.In particular, we identify three strategies for solving LWE, and give the algorithms available in the literature for solving LWE via one of these strategies. While in recent years several such algorithms were proposed and analysed, most treatments of LWE do not consider these results when discussing its hardness. By providing an accessible survey on available techniques we hope to motivate research to push the state-of-the-art in this area forward.

We note that in most previous works the hardness of LWE is treated only asymptotically. Indeed, it is not uncommon to hide logarithmic and constant factors in the exponent of complexity expressions. For example, Arora and Ge [AG11] specify the complexity of their algorithm as $2^{\tilde{\mathcal{O}}(n^{2\xi})}$, for some ξ such that $\alpha q = n^{\xi}$. While such statements – separating essential from inessential factors – allow us to understand the behaviour of various families of algorithms and of the problem in general, they need to be refined in order to gain insights into the concrete hardness of LWE.The importance of this could be seen, for example, when it comes to designing actual systems based on LWE.Here we must select parameters to ensure that the problem instance generated is hard with respect to a particular security parameter λ while still keeping parameters as small as possible for performance reasons. For this we must be able to identify the fastest known way of solving LWE with that choice of parameters, and be assured that this attack takes 2^{λ} operations. The second contribution of this survey is hence that where possible we provide concrete estimates for how long it takes to solve LWE.

Since for most algorithms no closed formulae are known expressing their complexities, the third contribution of this survey is that we provide a module for the Sage mathematics software [S+14] which, given the parameters of an LWE instance, outputs estimates for the concrete running time of the algorithms discussed in this survey. We also apply this estimator to various families of LWE parameters from the literature and discuss areas where the collective knowledge is limited in order to motivate further research.

Instances. To this end we need to characterise LWE instances. In this survey we always let χ be a discrete Gaussian distribution with *centre* zero and *width parameter* αq, denoted by $\mathcal{D}_{\mathbb{Z},\alpha q}$. A discrete Gaussian distribution with centre μ and width parameter αq samples elements with a probability proportional to $\exp(-\pi\frac{(x-\mu)^2}{(\alpha q)^2})$. The standard deviation of a continuous Gaussian with width parameter αq is $\sigma = \frac{\alpha q}{\sqrt{2\pi}}$ and we roughly have this relation when we discretise, as long as σ is bigger than the smoothing parameter $\eta_\epsilon(\mathbb{Z})$ of $\mathbb{Z}$ [DB13]. For ease of analysis, some works (e.g. [LP11]) treat the error terms as not too dissimilar from samples from a continuous Gaussian, and we join them in this approach whenever this occurs.

We then characterise LWE instances as follows:

1. Typically, we have $q \approx n^c$ and $\alpha q = \sqrt{n}$, i.e. $\alpha \approx n^{1/2-c}$, for c a small constant. The relation $\alpha q > \sqrt{n}$ allows the reduction of GapSVP to LWE to go through [Reg09]. In particular, Regev uses $\alpha q = 2\sqrt{n}$. Intuitively this is because a step in the reduction loses a factor of $\sqrt{n}$. Furthermore, if $\alpha q < \sqrt{n}$ then Arora and Ge's algorithm is subexponential [AG11]. In this survey, we simply pick $\alpha q = \sqrt{n}$, ignoring the constant 2 as it does not affect our estimates much. In this case we may characterise the instance by n (and c).
2. The most generic characterisation is by n, α, q.
3. In some applications, the secret $\mathbf{s}$ is not chosen uniformly at random from $\mathbb{Z}_q$ but we have the guarantee that all the $\mathbf{s}_{(i)}$ are "small", e.g. $\in \{0, 1\}$. In this case we characterise the instance by n, α, q, ψ where ψ is the distribution of the $\mathbf{s}_{(i)}$.

In many applications, we are only given access to $m = \tilde{\mathcal{O}}(n)$ samples. In this case, we would characterise the instance by m, n, α, q. However, in this work we will assume that we have access

to as many samples m as we require. This is a reasonable assumption because the hardness of the LWE problem itself is essentially independent of the number of samples [Reg10]. This could be explained by the result that given a fixed (polynomial) number of samples, one can generate arbitrarily many more, with only a slight worsening in the error [GPV08,ACPS09]. The new samples come with the caveat, though, that they are no longer independent.

Structure. In Section 2 we give relevant tools which we will use later. In Section 3 we review lattice reduction algorithms as these will also be useful. In Section 4 we explain the three main strategies for solving LWE.In Section 5 we describe the algorithms which can be used to solve LWE via a chosen strategy. In particular we consider instances of LWE characterised both by n, α, q and the special case $q = n^c$, $\alpha q = \sqrt{n}$. In Section 6 we concentrate on the third characterisation of LWE:those instances with a small secret. In Section 7 we apply our estimator to parameter choices from the literature. Finally, in Section 8 we make some concluding remarks.

2 Notation & Tools

Logarithms are base 2 if not stated otherwise. We write ln for the natural logarithm. We denote vectors in bold, e.g. $\mathbf{a}$, and matrices in upper-case bold, e.g. $\mathbf{A}$. By $\mathbf{a}_{(i)}$ we denote the i-th component of $\mathbf{a}$, i.e. a scalar. In contrast, $\mathbf{a}_i$ is the i-th element of a list of vectors. We denote by $\langle \cdot, \cdot \rangle$ the usual dot product of two vectors and by $\langle \cdot, \cdot \rangle_p$ this dot product modulo p. We write $2 \leq \omega < 3$ for the linear algebra constant. We adopt the convention that the first non-zero vector, say $\mathbf{b}_0$, in a reduced lattice basis is the shortest vector in the basis.

Since we will use lattice reduction later on, we need some basic definitions about lattices. A lattice L in $\mathbb{R}^m$ is a discrete additive subgroup. In this survey we restrict our attention to viewing a lattice $L(\mathbf{B})$ as being generated by a (non-unique) basis $\mathbf{B} = \{\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}\} \subset \mathbb{Z}^m$ of linearly-independent integer vectors. The rank of the lattice L is defined to be the rank of the basis matrix $\mathbf{B}$ with rows consisting of the basis vectors. If the rank equals m we say that L is full-rank. We are only concerned with such lattices in this work and henceforth assume that the lattices we deal with are full-rank. In addition, we are only concerned with q-ary lattices which are those such that $q\mathbb{Z}^m \subseteq L \subseteq \mathbb{Z}^m$. Note that every q-ary lattice is full-rank. Throughout, we adopt the convention that a lattice is generated by integer combinations of row vectors, to match software conventions. The volume $\mathrm{vol}\,(L)$ of a full-rank lattice L is the absolute value of the determinant of any basis of the lattice. The i^{th} *successive minimum* of a lattice, $\lambda_i(L)$, is the radius of the smallest ball centred at the origin containing at least i linearly independent lattice vectors. The *Gaussian heuristic* states that $\lambda_1(L) \approx \sqrt{\frac{m}{2\pi e}} \mathrm{vol}\,(L)^{1/m}$.

We now give three lemmas which will be useful later. The first shows that given samples from $L_{\mathbf{s},\chi}$ we can construct LWE instances where the secret vector follows the same distribution as the error.

Lemma 1 ([ACPS09]). *Let $\mathcal{D}_{\mathbb{Z}^n,\alpha q}$ be an n-dimensional extension of $\mathcal{D}_{\mathbb{Z},\alpha q}$ where each component is sampled according to $\mathcal{D}_{\mathbb{Z},\alpha q}$. Then, given access an oracle $L_{\mathbf{s},\chi}$ returning samples of the form $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ with $\mathbf{a} \leftarrow \mathcal{U}\left(\mathbb{Z}_q^n\right)$, $e \leftarrow \mathcal{D}_{\mathbb{Z},\alpha q}$ and $\mathbf{s} \in \mathbb{Z}_q^n$, we can construct samples of the form $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{e} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ with $\mathbf{a} \leftarrow \mathcal{U}\left(\mathbb{Z}_q^n\right)$, $e \leftarrow \mathcal{D}_{\mathbb{Z},\alpha q}$ and $\mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\alpha q}$ in $2\,n^2$ operations in $\mathbb{Z}_q$ per sample, at the loss of n samples overall and with $\mathcal{O}\left(n^\omega\right)$ operations for precomputation.*

Proof. Take n samples from $L_{\mathbf{s},\chi}$ and write:

$$(\mathbf{A}_0, \mathbf{c}_0) = (\mathbf{A}_0, \mathbf{A}_0 \cdot \mathbf{s} + \mathbf{e}_0)$$

where $\mathbf{A}_0 \in \mathbb{Z}_q^{n \times n}$. With probability $\prod_{i=1}^{n}(q^n - q^{i-1})/q^{n^2}$ this matrix is invertible. Precompute $\mathbf{A}_0^{-1}$ and store it; this costs $\mathcal{O}(n^\omega)$ operations. Now, to produce n samples of the form $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{e} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ with $\mathbf{a} \leftarrow \mathcal{U}\left(\mathbb{Z}_q^n\right)$, $e \leftarrow \mathcal{D}_{\mathbb{Z},\alpha q}$ and $\mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\alpha q}$ we sample:

$$(\mathbf{a}_1, \mathbf{c}_1) = (\mathbf{a}_1, \mathbf{a}_1 \cdot \mathbf{s} + \mathbf{e}_1)$$

from $L_{\mathbf{s},\chi}$ and compute:

$$\begin{aligned}
\mathbf{a}_1 \cdot \mathbf{A}_0^{-1} \cdot \mathbf{c}_0 - \mathbf{c}_1 &= \mathbf{a}_1 \cdot \mathbf{A}_0^{-1}(\mathbf{A}_0 \cdot \mathbf{s} + \mathbf{e}_0) - \mathbf{a}_1 \cdot \mathbf{s} - \mathbf{e}_1 \\
&= \mathbf{a}_1 \cdot \mathbf{A}_0^{-1} \cdot \mathbf{A}_0 \cdot \mathbf{s} + \mathbf{a}_1 \cdot \mathbf{A}_0^{-1}\mathbf{e}_0 - \mathbf{a}_1 \cdot \mathbf{s} - \mathbf{e}_1 \\
&= \mathbf{a}_1 \cdot \mathbf{s} + \mathbf{a}_1 \cdot \mathbf{A}_0^{-1}\mathbf{e}_0 - \mathbf{a}_1 \cdot \mathbf{s} - \mathbf{e}_1 \\
&= \mathbf{a}_1 \cdot \mathbf{A}_0^{-1}\mathbf{e}_0 - \mathbf{e}_1.
\end{aligned}$$

Now, since $\mathcal{D}_{\mathbb{Z},\alpha q}$ is symmetric and $\mathbf{A}_0^{-1}$ has full rank, we get that

$$(\mathbf{a}_1 \cdot \mathbf{A}_0^{-1}, \mathbf{a}_1 \cdot \mathbf{A}_0^{-1}\mathbf{c}_0 + \mathbf{c}_1) = (\mathbf{a}_1 \cdot \mathbf{A}_0^{-1}, \mathbf{a}_1 \cdot \mathbf{A}_0^{-1}\mathbf{e}_0 - \mathbf{e}_1)$$

are n valid samples of the form $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{e} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ with $\mathbf{a} \leftarrow \mathcal{U}\left(\mathbb{Z}_q^n\right)$, $e \leftarrow \mathcal{D}_{\mathbb{Z},\alpha q}$ and $\mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z}^n,\alpha q}$. Finally, computing $\mathbf{a}_1 \cdot \mathbf{A}_0^{-1}$ takes $2\,n^2$ operations in $\mathbb{Z}_q$. □

In certain cases we have the notion of 'modulus switching' which means that given samples from $L_{\mathbf{s},\chi}$, we can construct LWE instances where the modulus is now p for some particular $p < q$. Modulus switching was initially introduced to speed-up homomorphic encryption [BV11] but can also be employed to reduce the cost of solving LWE in certain cases [AFFP14]. Modulus switching can be thought of as analogous to the difference between computing with single instead of double precision floating point numbers, where switching refers to opting to compute in the lower precision of a machine float. In the LWE context, for some $p < q$, modulus switching is considering an instance of LWE (mod q) as a scaled instance of LWE (mod p). This incurs a noise increase which is only small if $\mathbf{s}$ is small, so the technique can only be used for small secrets. The requirements on p must be balanced. On the one hand, minimising p will minimise the running time of most algorithms (see Section 6). On the other hand, picking p too small increases the noise level leading to a higher solving complexity. Our choice of p ensures that $\left\| \left\langle \frac{p}{q} \cdot \mathbf{a} - \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle \right\| \approx \frac{p}{q} \cdot \|e\|$ if $\mathbf{s}$ is small enough. This means that the new error term after modulus switching is essentially the previous error scaled. In particular, we have the following lemma:

Lemma 2 ([BV11,BLP+13]). *Let $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ be sampled from $L_{\mathbf{s},\chi}$. Let $p \approx \sqrt{\frac{2\pi\, n}{12}} \cdot \frac{\sigma_s}{\alpha}$, where σ_s is the standard deviation of elements in the secret $\mathbf{s}$. If $p < q$ then*

$$\left(\left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \left\lfloor \frac{p}{q} \cdot c \right\rceil \right) \text{ in } \mathbb{Z}_p^n \times \mathbb{Z}_p$$

follows a distribution close to $L_{\mathbf{s},\mathcal{D}_{\mathbb{Z},\sqrt{2}\alpha p + \mathcal{O}(1)}}$.

Proof. Consider

$$\begin{aligned}
\left\lfloor \frac{p}{q} \cdot c \right\rceil &= \left\lfloor \frac{p}{q} \left(\langle \mathbf{a}, \mathbf{s} \rangle + e \right) \right\rceil \\
&= \left\lfloor \left\langle \frac{p}{q} \cdot \mathbf{a}, \mathbf{s} \right\rangle_p + \frac{p}{q} \cdot e \right\rceil \\
&= \left\lfloor \left\langle \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + \left\langle \frac{p}{q} \cdot \mathbf{a} - \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + \frac{p}{q} \cdot e \right\rceil \\
&= \left\langle \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + \left\langle \frac{p}{q} \cdot \mathbf{a} - \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + \frac{p}{q} \cdot e + e', \text{ where } e' \in [-0.5, 0.5] \\
&= \left\langle \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + e'' + \frac{p}{q} \cdot e + e'.
\end{aligned}$$

Since $\frac{p}{q} \cdot \mathbf{a} - \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil$ takes values $\in [-0.5, 0.5]$ we have that $e'' \approx \sqrt{n/12}\,\sigma_s$. Targeting $e'' \approx \frac{p}{q} \cdot e$ we get

$$\begin{aligned}
\frac{p}{q} \cdot \alpha q / \sqrt{2\pi} &\approx \sqrt{n/12}\,\sigma_s \\
p \cdot \alpha / \sqrt{2\pi} &\approx \sqrt{n/12}\,\sigma_s \\
p &\approx \frac{\sqrt{2\pi}}{\alpha} \cdot \sqrt{n/12}\,\sigma_s. \\
p &\approx \sqrt{\frac{2\pi\, n}{12}} \cdot \frac{\sigma_s}{\alpha}.
\end{aligned}$$

$\frac{p}{q} \cdot e$ is a scaled discrete Gaussian, e'' is an inner product and thus approaches a discrete Gaussian as n increases, e' is distributed in $[-0.5, 0.5]$ from which the claim follows. □

The following lemma shows the equivalence of Decision-LWE and Search-LWE. Search to decision is trivial: if search is solved, $\mathbf{s}$ is known, so $e = c - \langle \mathbf{a}, \mathbf{s} \rangle$ can be computed. The nontrivial direction is due to Regev [Reg09], which is reproduced with proof below. Having established equivalence, whenever a method can be shown to solve Search-LWE or Decision-LWE we can speak of it solving LWE.

Lemma 3 (Lemma 4.2 in [Reg09]). *Let $n \geq 1$ be some integer, $2 \leq q \leq \text{poly}(n)$ be a prime, and χ be some distribution on $\mathbb{Z}_q$. Assume that we have access to a procedure W that, for all $\mathbf{s}$, accepts with probability exponentially close to 1 on inputs from $L_{\mathbf{s},\chi}$ and rejects with probability exponentially close to 1 on uniformly random inputs. Then, there exists an efficient algorithm W' that, given samples from $L_{\mathbf{s},\chi}$ for some $\mathbf{s}$, outputs $\mathbf{s}$ with probability exponentially close to 1.*

Proof. We show how W' finds the first component $\mathbf{s}_{(0)}$ of $\mathbf{s}$; finding the other components is similar. For any $k \in \mathbb{Z}_q$ consider the following transformation. Given a pair $(\mathbf{a}, c)$ as input to W', let it output the pair $(\mathbf{a} + (l, 0, \ldots, 0), c + lk)$ where $l \in \mathbb{Z}_q$ is chosen uniformly at random. It is easy to see that this transformation takes the uniform distribution to itself. On the other hand suppose the input pair $(\mathbf{a}, c)$ is sampled from $L_{\mathbf{s},\chi}$. If $k = \mathbf{s}_{(0)}$ then this transformation takes $L_{\mathbf{s},\chi}$ into itself. If $k \neq \mathbf{s}_{(0)}$ then this transformation takes $L_{\mathbf{s},\chi}$ to the uniform distribution. There are only polynomially many (namely q) possibilities for $\mathbf{s}_{(0)}$, so we can try all of them as possible k values. For each k value, let the output of W' be the input to W. Then as W can distinguish $L_{\mathbf{s},\chi}$ from uniform, it can tell whether $k = \mathbf{s}_{(0)}$. □

3 Lattice Reduction Algorithms

Many algorithms for solving LWE rely on lattice reduction as the central step. Hence, in this section we briefly review lattice reduction algorithms and discuss the current state-of-affairs in terms of estimating their running time. Since this survey is concerned with discussing algorithms for solving LWE this section is kept rather brief and the interested reader is directed to [MR09,Reg09,Ngu10,Ngu11,LP11] for further details on lattices.

Lattice reduction algorithms can be viewed as a hierarchy: cases of BKZ based on the block size parameter k. For $k = 2$ we are essentially running LLL, which runs in polynomial time but the reduced basis output will only contain a short vector to within exponential factors. When $k = n$, i.e the full size of the basis, then the output basis would be HKZ (Hermite-Korkine-Zolotarev) reduced. This is in some sense optimally reduced, but requires exponential runtime. Hence, when performing lattice reduction, one generally uses BKZ with some intermediary block size.

The quality of a basis output by a lattice reduction algorithm is characterised by the Hermite factor δ_0^n, which is defined such that the shortest non-zero vector $\mathbf{b}_0$ in the output basis has the following property: $\|\mathbf{b}_0\| = \delta_0^n \operatorname{vol}(L)^{1/n}$. We may also refer to δ_0 itself, and call it the root-Hermite factor. We call its logarithm to base 2 the log root-Hermite factor.

3.1 LLL

LLL can be considered as a generalisation of the two dimensional algorithm by Lagrange and sometimes attributed to Gauss (see for example [Jou09]). The output of this algorithm is a basis $\{\mathbf{b}_0, \mathbf{b}_1\}$ such that $\|\mathbf{b}_0\| \leq \|\mathbf{b}_1\|$ and the Gram-Schmidt coefficient $\mu_{1,0} \leq \frac{1}{2}$. In particular, $\|\mathbf{b}_0\| = \lambda_1$ and $\|\mathbf{b}_1\| = \lambda_2$. The algorithm works by taking in a pair of vectors $\mathbf{b}_0 \leq \mathbf{b}_1$ and then setting $\mathbf{b}_1 = \mathbf{b}_1 - \lfloor \mu_{1,0} \rceil \mathbf{b}_0$, then swapping the vectors and repeating until no more changes can be made. Thus, when this terminates, we must have $\mu_{1,0} \leq \frac{1}{2}$.

To extend into higher dimensions one would like to do something similar but the optimal way to do this is not clear because of the additional choice of directions. Notice that the Gauss algorithm ensures that $\frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_0\|}$ is not too small, in particular, $\frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_0\|} \geq 1 - \mu_{1,0}^2 \geq \frac{3}{4}$ (see e.g. [LvdPdW12] for more details). An LLL reduced basis satisfies a relaxed general version of this: $\frac{\|\mathbf{b}_i\|}{\|\mathbf{b}_{i-1}\|} \geq \delta - \mu_{i,i-1}^2$ for some $\delta \in \left(\frac{1}{4}, 1\right)$. This relaxation, known as the Lovász condition, is necessary for polynomial runtime. A typical choice is $\delta = \frac{3}{4}$.

More formally, a basis $\{\mathbf{b}_0, \mathbf{b}_1, \ldots \mathbf{b}_{n-1}\}$ is LLL-reduced if it satisfies the Lovász condition (for some δ) and it is size reduced; that is, $\mu_{i,j} \leq \frac{1}{2}$ for $0 \leq j < i \leq n - 1$.

Essentially, LLL works by size reducing the basis vectors pairwise, and then checking if the Lovász condition still holds; if it does not, then it swaps the current vector with the previous vector. In more detail, let the input basis be $\{\mathbf{b}_0, \ldots \mathbf{b}_{n-1}\}$. Starting at $i = 1$ and incrementing upwards, consider $\mathbf{b}_i$ and size reduce with respect to $\mathbf{b}_j$ for $j = i - 1$ down to $j = 0$. Then check if $\mathbf{b}_i$ and $\mathbf{b}_{i-1}$ satisfy the Lovász condition. If they do, increment i; if not, swap them and decrement i to ensure the swap has not affected the Lovász condition holding in the previous pair.

Running Time. It is well known the runtime of LLL is polynomial and indeed this was proved as it was introduced [LLL82]. In particular for an input basis where for all i, $\|\mathbf{b}_i\| < B$, LLL outputs an LLL-reduced basis in time $\mathcal{O}\left(n^{5+\varepsilon} \log^{2+\varepsilon} B\right)$ (using fast integer multiplication). In more recent variants, improvements have been made. For example, one variant introduced by Nguyen and Stehlé called L2 [NS05] provably outputs an LLL-reduced basis in time $\mathcal{O}\left(n^{5+\varepsilon} \log B + n^{4+\varepsilon} \log^2 B\right)$

(using fast integer multiplication). That is, one that only grows quadratically in $\log B$. Heuristically, variants of LLL achieve $\mathcal{O}\left(n^3 \log^2 B\right)$ [CN11].

Quality of Output. LLL theoretically achieves a Hermite factor of $\left(\frac{4}{3}\right)^{\frac{n-1}{4}}$ [LLL82]. In practice, it behaves much better and a root-Hermite factor δ_0 of 1.0219 is reported in [GN08].

Implementations. LLL and its variants is implemented in many software packages, notably in NTL [Sho], FLINT [HJP14] and fplll [CPS13]. The latter also implements L2.

3.2 BKZ

The BKZ algorithm requires an algorithm solving exact SVP in possibly smaller dimensions as a subroutine. The typical methods of doing this are computing the Voronoi cell of the lattice, sieving or enumeration [HPS11b]. Below we refer to running any of these algorithms as calling an SVP oracle.

The BKZ algorithm runs as follows, where at every stage $\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}$ is the updated basis. The input basis is LLL reduced, and the first block is $\mathbf{b}_0, \ldots, \mathbf{b}_{k-1}$. Call the SVP oracle to obtain a short vector, $\mathbf{b}_0^*$, in the space spanned by these vectors. We now have $k+1$ vectors spanning a k dimensional space, so we call LLL to obtain a new set of k linearly independent vectors. The second block is made of vectors which are the projection of $\mathbf{b}_1, \ldots, \mathbf{b}_k$ onto $\langle \mathbf{b}_0 \rangle^\perp$ (the space which is the span of the orthogonal complement of $\mathbf{b}_0$). Again we call the SVP oracle to obtain a short vector in this space, $\mathbf{b}_1^*$, which can be viewed as the projection of some $\mathbf{b}_1'$ in the lattice. Now we call LLL on $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_k, \mathbf{b}_1'$ to update the list of basis vectors. The next block is made of vectors which are the projection of $\mathbf{b}_2, \ldots, \mathbf{b}_{k+1}$ onto $\langle \mathbf{b}_0, \mathbf{b}_1 \rangle^\perp$ (the space which is the span of the orthogonal complement of $\mathbf{b}_0$ and $\mathbf{b}_1$), and again the SVP oracle is called to obtain a short vector in this space, which can be viewed as a projected $\mathbf{b}_2'$; and this procedure carries on through the basis. The first $n-k+1$ blocks are all of size k, and then after this point each block is one vector shorter than the previous block. The output basis of this process is another LLL reduced basis, which can be treated as a new input, and the whole process continues again, until a basis passes through unchanged, at which point the algorithm terminates.

A HKZ reduced basis $\{\mathbf{b}_0, \ldots \mathbf{b}_{d-1}\}$ is a basis such that its Gram-Schmidt vectors $\mathbf{b}_i^*$ satisfy the property that $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(L))$ for $0 \leq i \leq d-1$ where $\pi_i(L) = \langle \mathbf{b}_0, \ldots, \mathbf{b}_{i-2} \rangle^\perp$. We can see that BKZ constructively achieves a basis with the following property: each block of size k (e.g. $\mathbf{b}_0, \ldots, \mathbf{b}_{k-1}$), that is all the first $n-k+1$ blocks, is a HKZ reduced basis. Therefore, if $k = n$ then the whole output basis is HKZ reduced.

BKZ 2.0. Several improvements of BKZ have been suggested and their combination is often referred to as BKZ 2.0 [CN11]. These improvements include some heuristically better techniques such as extreme pruning [GNR10], early termination, and local block pre-processing. Extreme pruning takes place in the enumeration subroutine, and it works by not exploring all branches in the search tree, with the hope that a short enough vector is still found, therefore decreasing runtime. Early termination is based on the heuristic observation that the quality of the output basis increases more dramatically in the earlier rounds of BKZ. Therefore, continuing to reduce the lattice offers diminishing returns in the basis quality, and early termination decreases the runtime while still returning a basis close to the desired quality. Local block pre-processing takes the form of running BKZ-k' with early termination for some value k' so that the local basis is more than merely LLL reduced.

Quality of Output. Chen [Che13] gives a limiting value of δ_0 achievable by BKZ as a function of the block size k which we may apply as an estimate for δ_0 even when n is finite:

$$\lim_{n\to\infty} \delta_0 = \left(\frac{k}{2\pi e}(\pi k)^{\frac{1}{k}}\right)^{\frac{1}{2(k-1)}}. \tag{1}$$

The same numerical estimate can also be achieved by running the BKZ 2.0 simulator [CN11].

The 'lattice rule of thumb' is often used to approximate what δ_0 a given k can achieve by $\delta_0 = k^{\frac{1}{2k}}$. This expression, in turn is often approximated by $\delta_0 = 2^{1/k}$ [Ste13] to ease analysis.

We note that depending on which estimate is used vastly different relations between k and δ_0 are assumed. To illustrate this, we plot predictions for δ_0 for block sizes $50 \leq k \leq 250$ in Figure 1.

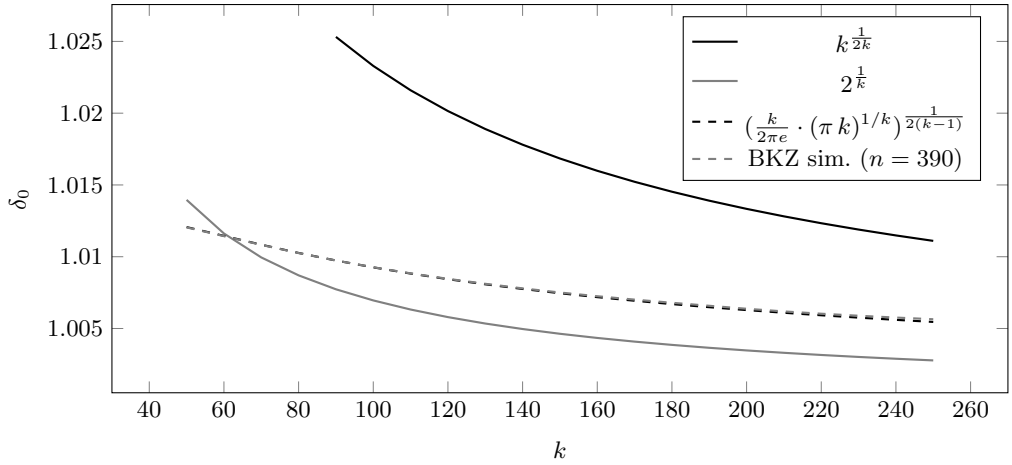

Fig. 1. Estimates for δ_0 for BKZ-k. The BKZ simulator was run on input resembling a 390×390 dimensional q-ary lattice.

Assuming that the data from the BKZ simulator is a good enough indication for actual behaviour, we may conclude from Figure 1 that we do not need to consider the approximation $k^{\frac{1}{2k}}$ as it is always too pessimistic. The approximation $2^{\frac{1}{k}}$ is closer to the actually expected behaviour, but as we will show below it implies a simple sub-exponential algorithm for solving LWE via straightforward lattice reduction.

Running Time. The running time of BKZ is mainly determined by two factors: firstly, the time t_k it takes to find shortest or short enough vectors in lattices of dimension k; and secondly, the number of BKZ rounds ρ needed. We assume CPU clock cycles as our basic unit to abstract from CPU clock speeds. If t_k is the number of clock cycles it takes to solve SVP in dimension k we expect BKZ to take $\rho \cdot n \cdot t_k$ clock cycles.

SVP Oracles. As mentioned above, three main families of algorithms exist for finding shortest vectors [HPS11b]. Computing the Voronoi cell of the lattice takes about $2^{2k+o(k)}$ operations and $2^{k+o(k)}$ memory. Sieving takes about $2^{2.465k+o(k)}$ operations and $2^{1.325k+o(k)}$ memory in its provable variant and $2^{0.3774\,k}$ operations and $2^{0.2925\,k}$ memory in its heuristic variant [BGJ13]. Enumeration is typically implemented in a fashion requiring $2^{\mathcal{O}(k^2)}$ operations and poly(k) memory, but can

be shown to achieve $k^{\mathcal{O}(k)}$ operations and poly(k) memory (cf. [MW14] for a recent proposal with low overhead). All known implementations of BKZ use enumeration with a complexity of $2^{\mathcal{O}(k^2)}$ as it beats other methods for dimensions achievable in practice. We note that this complicates deriving estimates for large block sizes from experimental evidence.

Estimating ρ. No closed formula for the expected number of BKZ rounds is known. The best upper bound is exponential, but after $\rho \approx \frac{n^2}{k^2} \log n$ many rounds, the quality of the basis is already very close to the final output [HPS11a].

Asymptotic Behaviour. Before we discuss existing estimates for the running time of BKZ in the literature, we briefly discuss the expected asymptotic behaviour of the algorithm. The 'lattice rule of thumb' puts the relation between the block size k and δ_0 as $\delta_0 = k^{1/2k}$, which implies $k/\log(k) = 1/(2\log\delta_0)$. To solve this for k we need the following technical lemma:

Lemma 4. *For $i \geq 1$, let $g_i(x) = x\log(g_{i-1}(x))$ with $g_0(x) = 2$. If $a/\log(a) = b$ and $\log(a) \geq 1$ then*

$$a \geq g_n(b)$$

for any $n \geq 0$. In particular, for $\log(a) > 2$, $a = g_\infty(b)$.

Proof. For the first claim, notice that $a \geq g_0(b) = 2$ as $\log(a) \geq 1$. Furthermore, $a \geq g_1(b) = b$ as $a/\log(a) = b$ so $a \geq b$. We also have $a \geq g_2(b) = b\log(b)$:

$$\begin{aligned} a &= b\log a \\ \Rightarrow\ a &\geq b \\ \Rightarrow \log a &\geq \log b \\ \Rightarrow\ a &\geq b\log b \end{aligned}$$

For the inductive step,

$$\begin{aligned} \text{suppose}\quad a &\geq g_i(b) \\ \Rightarrow \log a &\geq \log(g_i(b)) \\ a &= b\log a \\ \Rightarrow\ a &\geq b\log(g_i(b)) = g_{i+1}(b). \end{aligned}$$

So by induction, we have $a \geq g_n(b)$. For the second claim, when $\log(a) > 2$, $b = a/\log(a) > 2$. We now prove by induction that $g_n(b) \geq g_{n-1}(b)$ for all $n \geq 1$. For the base case, we have $g_1(b) = b > 2 = g_0(b)$. For the inductive step,

$$\begin{aligned} \text{suppose}\quad g_i(b) &\geq g_{i-1}(b) \\ \Rightarrow\ \frac{g_i(b)}{g_{i-1}(b)} &\geq 1 \\ \Rightarrow\ \log\left(\frac{g_i(b)}{g_{i-1}(b)}\right) &\geq 0 \\ \text{Now } g_{i+1}(b) - g_i(b) = b\log\left(\frac{g_i(b)}{g_{i-1}(b)}\right) &\geq 0 \\ \Rightarrow\ g_{i+1}(b) &\geq g_i(b) \end{aligned}$$

Thus we have that $g_n(b)$ is an increasing sequence, and by the first claim, it is bounded above by a. So it is convergent and we may denote its limit by $g_\infty(b)$. This satisfies $g_\infty(b) = b\log(g_\infty(b))$

and so $\frac{g_\infty(b)}{\log(g_\infty(b))} = \frac{a}{\log(a)}$. Now, for $x \geq 4$, the function $\frac{x}{\log(x)}$ is one-to-one. Note that we have $g_n(b) > 2$ for all n so also $g_\infty(b) > 2$.

It remains to prove that $g_\infty(b) \geq 4$, which implies $g_\infty(b) = a$. To show this, we require some further properties of the function $\frac{x}{\log(x)}$. Consider the solutions of the equation $\frac{x}{\log(x)} = 2$. These are precisely $x = 2$ and $x = 4$. By differentiating $\frac{x}{\log(x)}$ and evaluating at these values, and with the observation that $\frac{x}{\log(x)}$ is continuous for $x > 1$, we can see that $\frac{x}{\log(x)}$ takes values below 2 precisely for $2 < x < 4$. But, $a/\log(a) > 2$. So we must be in the region $x \geq 4$. So, $\frac{x}{\log(x)}$ is injective here and we may conclude $g_\infty(b) = a$ as required. □

Hence, we have $k \geq g_n\left(\frac{1}{2\log\delta_0}\right)$. In particular, we have $k \geq g_1\left(\frac{1}{2\log\delta_0}\right) = \frac{-\log(2\log\delta_0)}{2\log\delta_0}$. Ignoring constants, this expression simplifies to $\frac{-\log(\log\delta_0)}{\log\delta_0}$. It follows that the log of the time complexity of the BKZ algorithm is as follows.

Lemma 5. *The log of the time complexity for running BKZ to achieve a root-Hermite factor δ_0 is:*

$$\begin{cases} \Omega\left(\frac{\log^2(\log\ \delta_0)}{\log^2\delta_0}\right) & \text{if calling the SVP oracle costs } \mathcal{O}\left(2^{n^2}\right), \\ \Omega\left(\frac{-\log\left(\frac{-\log\log\delta_0}{\log\delta_0}\right)\log\log\delta_0}{\log\delta_0}\right) & \text{if calling the SVP oracle costs } \mathcal{O}\left(n^n\right), \\ \Omega\left(\frac{-\log\log\delta_0}{\log\delta_0}\right) & \text{if calling the SVP oracle costs } \mathcal{O}\left(2^n\right). \end{cases}$$

Existing Estimates. The following estimates for the running time of BKZ exist in the literature.

– Lindner and Peikert [LP11] give an estimate for the runtime (in seconds) of BKZ as

$$\log t_{BKZ}(\delta_0) = \frac{1.8}{\log\delta_0} - 110$$

based on experiments with the implementation of BKZ in the NTL library [Sho]. That is, improvements such as extreme pruning, early termination, and local block pre-processing were not used. To convert the estimate to a more general metric, we may notice that it was derived from experiments performed on a computer running at 2.3GHz. We can hence convert this to clock cycles, giving a runtime of $2^{\frac{1.8}{\log\delta_0}-110} \cdot 2^{\log(2.3\cdot10^9)} = 2^{\frac{1.8}{\log\delta_0}-110+\log(2.3\cdot10^9)} \approx 2^{\frac{1.8}{\log\delta_0}-78.9}$ clock cycles. It should be noted that this is a linear model, which does not fit the actual implementation on BKZ in the NTL library as this uses an enumeration subroutine requiring $2^{\mathcal{O}(k^2)}$ time. As we will show below in Section 5.3, applying this model to predict the behaviour of BKZ leads to a subexponential algorithm for LWE, which is widely believed to not be the case.

– Albrecht et al. [ACF+13] use data points of Liu and Nguyen [LN13] to extrapolate a model similar to Lindner and Peikert's [LP11] and conclude the running time of BKZ 2.0 (in seconds) to be

$$\log t_{BKZ}(\delta_0) = \frac{0.009}{\log^2\delta_0} - 27.$$

They argue that for current implementations and estimates based on them the runtime of BKZ being nonlinear in $\log\delta_0$ is more fitting than a linear model such as that of Lindner

and Peikert. The analysis is in the same metric, in particular it gives a runtime in seconds assuming a 2.3GHz computer, so we can convert this into clock cycles to give a runtime of $2^{\frac{0.009}{\log^2 \delta_0} - 27 + \log(2.3 \cdot 10^9)} \approx 2^{\frac{0.009}{\log^2 \delta_0} + 4.1}$. We refer to this as the delta-squared model. It should be noted, though, that the running times on which this model is based were not independently verified which limits their utility. Note that this estimate drops the $\log^2(\log \delta_0)$ factor from Lemma 5.

- Chen and Nguyen provide an simulation algorithm for BKZ 2.0 [CN11,Che13] for arbitrarily high block size, under the assumption that each block behaves as a random basis. The authors note that this assumption may not hold for block sizes $k < 50$. The algorithm takes as input the logs of the norms of the Gram-Schmidt vectors belonging to the input matrix and a block size k. It outputs the expected logs of the norms of the Gram-Schmidt vectors of the BKZ-k reduced basis as well as the number of BKZ rounds ρ needed. Combined with estimates for t_k (see below) we may turn this into an estimator for the running time.
- van de Pol and Smart [vdPS13] consider the problem from the perspective of using BKZ to solve an LWE or lattice-based system. They assume one has a desired level of security 2^λ (a maximum number of operations an adversary can perform) and a lattice dimension m and using this to find the lowest δ_0 which can be achieved in 2^λ operations, minimising over possible choices of the block size k and the number of rounds $\rho = \rho(k, m, \lambda)$. This is in contrast to an approach where the parameters of the system correspond to a δ_0 which then implies a certain security level. They use a table of Chen and Nguyen [CN11, Table 3] to estimate the cost of one enumeration for a given k and to calculate the total number of enumerations one can perform for this k (to reach the maximum of 2^λ operations). Note that this means they do not consider block sizes $k > 250$ as Chen and Nguyen do not give estimates for those. Smart and van de Pol remark that δ_0 seems to converge to a value depending only on k, corroborating other results in the literature. They note further that the convergence is slower in higher dimension.

Estimates for t_k. In Table 1 we list estimates for solving SVP in dimension k which were derived as follows. The first row – labelled 'fplll' – was derived by calling the SVP function available in fplll 4.0.4 [CPS13] for dimensions up to 53 and by fitting $ak^2 + bk + c$ to the logs of these averaged running times. The second row – labelled 'enum' – was derived by fitting $ak^2 + bk + c$ to the last row of Table 3 in [CN11] and assuming one enumeration costs 200 clock cycles as in [CN11]. The third row is taken from [BGJ13] which reports a complexity of $0.3374\,k$ and a running time of 2080 hours for block size 90.

name	data source	$\log(t_k)$
fplll	fplll 4.0.4	$0.0135\,k^2 - 0.2825\,k + 21.02$
enum	[CN11]	$0.0029\,k^2 - 0.1227\,k + 23.83$
sieve	[BGJ13]	$0.3774\,k + 20$

Table 1. Estimates for the cost in clock cycles to solve SVP in dimension k.

Overall. By setting $\rho \approx \frac{n^2}{k^2}\log(n)$, we assume that running BKZ for block size k and dimension n costs $\frac{n^3}{k^2}\log(n) \cdot t_k$ CPU cycles where t_k is taken from Table 1 based on how the SVP oracle is instantiated.

Implementations. BKZ is implemented in NTL [Sho] and fplll [CPS13]. Neither of these implementations incorporate all techniques which are collectively known as BKZ 2.0. However, the next version of fplll is expected to make progress in that direction [ACPS14].

3.3 Choosing m

In some of the algorithms below we will have a choice of which lattice to consider. In particular, the situation will arise where our task is to find a vector with a target norm in a lattice with a given volume $\mathrm{vol}\,(L)$ but variable dimension. Given this degree of freedom, we will have to choose an optimal subdimension m to perform lattice reduction on. To find this optimal subdimension we need to find m such that

$$\|\mathbf{v}\| = \delta_0^m \,\mathrm{vol}\,(L)^{1/m}$$

is minimised. If, as in many applications below, $\mathrm{vol}\,(L) = q^n$ this becomes $\|\mathbf{v}\| = \delta_0^m \, q^{n/m}$. Then,

$$m = \sqrt{\frac{n \log q}{\log \delta_0}}$$

is the optimal subdimension to consider [MR09]. This 'optimal subdimension' is also often heuristically chosen even when the above relation between volume and dimension does not hold. In [vdPS13] the authors choose m based on the best δ_0 which can be obtained for a given security level. In one example the dimension they choose is similar to the 'optimal subdimension'.

4 Strategies

In this section we discuss three strategies for solving LWE: solving Decision-LWE by finding a short vector $\mathbf{v}$ such that $\langle \mathbf{v}, \mathbf{a} \rangle = 0$; solving Search-LWE by finding a short e such that $\langle \mathbf{a}, \mathbf{x} \rangle = c - e$ for some unknown $\mathbf{x}$; or solving Search-LWE by finding an $\mathbf{s}'$ such that $\langle \mathbf{a}, \mathbf{s}' \rangle$ is close to c. All algorithms in Section 5 follow one of these strategies.

4.1 Short Integer Solutions (SIS)

To distinguish the case where m samples $(\mathbf{A}, \mathbf{c})$ either: follow $L_{\mathbf{s},\chi}$, and hence satisfy $\mathbf{c} = \mathbf{A}\mathbf{s} + \mathbf{e}$ with $\mathbf{e}_{(i)} \leftarrow \mathcal{D}_{\mathbb{Z},\alpha q}$; or $\mathbf{c}$ is uniformly random, we can try to find a short vector $\mathbf{v}$ such that $\mathbf{v} \cdot \mathbf{A} = 0$. Expressed as a lattice problem, we aim to find a vector $\mathbf{v}$ in the scaled (by q) dual lattice of the lattice generated by $\mathbf{A}$, i.e. the lattice $L = \{\mathbf{w} \in \mathbb{Z}_q^m \mid \mathbf{w}\mathbf{A} \equiv 0 \mod q\}$, which is exactly solving the Short Integer Solutions problem. Consider $\langle \mathbf{v}, \mathbf{c} \rangle$. If $\mathbf{c} = \mathbf{A}\mathbf{s} + \mathbf{e}$ then $\langle \mathbf{v}, \mathbf{c} \rangle = \langle \mathbf{v}, \mathbf{e} \rangle$ which follows a Gaussian distribution over $\mathbb{Z}$ considered modulo q. In particular, it often returns small samples as both $\mathbf{v}$ and $\mathbf{e}$ are small. On the other hand, if $\mathbf{c}$ is uniform then $\langle \mathbf{v}, \mathbf{c} \rangle$ is uniform on $\mathbb{Z}_q$. So we may distinguish these two cases, thus solving Decision-LWE. We must however ensure $\|\mathbf{v}\|$ is suitably short. If $\|\mathbf{v}\|$ is too large then the (Gaussian) distribution of $\langle \mathbf{v}, \mathbf{e} \rangle$ will be too flat to distinguish from random. In particular, we have the following lemma:

Lemma 6 ([LP11]). *Given an LWE instance characterised by n, α, q and a vector $\mathbf{v}$ of length $\|\mathbf{v}\|$ in the scaled dual $L = \{\mathbf{w} \in \mathbb{Z}_q^m \mid \mathbf{w}\mathbf{A} \equiv 0 \mod q\}$, the statistical distance of $\langle \mathbf{v}, \mathbf{e} \rangle$ and the uniform distribution on $\mathbb{Z}_q$ is close to $\exp(-\pi \cdot (\|\mathbf{v}\| \cdot \alpha)^2)$.*

Remark 1. For example, Stehlé [Ste13] states that a suitably short choice to distinguish $L_{\mathbf{s},\chi}$ from random is $\|\mathbf{v}\| \cdot \alpha q \leq q$, i.e. $\|\mathbf{v}\| = 1/\alpha$. By Lemma 6, this results in a probability of about $1/23$ to distinguish correctly.

We note that depending on the algorithm used to obtain the short vector $\mathbf{v}$, it may be advantageous to accept a longer vector as output. This decreases the success probability ϵ, but then running the algorithm several (e.g $1/\epsilon$) times will achieve a success probability very close to 1. This may be faster than the alternative, which uses fewer vectors (runs of the algorithm) at a higher success probability, but takes significantly longer to obtain these shorter vectors

Corollary 1. *To obtain a probability ϵ of success in solving an LWE instance parametrised by n, q and α via the SIS strategy, we require a vector $\mathbf{v}$ with $\|\mathbf{v}\| = \frac{1}{\alpha}\sqrt{\ln(\frac{1}{\epsilon})/\pi}$.*

Methods of finding a short vector in the dual lattice, or in a lattice generally, will be described in the sections below. For ease of exposition we let $f(\epsilon)$ denote $\sqrt{\ln(\frac{1}{\epsilon})/\pi}$.

4.2 Bounded Distance Decoding (BDD)

Given m samples $(\mathbf{A}, \mathbf{c} = \mathbf{As} + \mathbf{e})$ following $L_{\mathbf{s},\chi}$ we may observe that $\mathbf{c}$ is close to a linear combination of the columns of $\mathbf{A}$. Furthermore, since the noise is Gaussian, almost all of the noise is within, say, three times the standard deviation (that is, $\frac{3\alpha q}{\sqrt{2\pi}}$) from 0. Consider the lattice spanned by the columns of $\mathbf{A}$. We can see that $\mathbf{c}$ is a point which is bounded in distance from a lattice point $\mathbf{w} = \mathbf{As}$. Hence, we may view the LWE instance as a Bounded Distance Decoding (BDD) problem instance in this lattice. This problem is as follows: given a basis of a lattice, a target vector, and a bound on the distance from the target to the lattice, find a lattice vector within that bound of the target vector. In this case, our solution to the BDD problem would be the lattice point $\mathbf{w}$, from which we may then use linear algebra to recover $\mathbf{s}$ and therefore solve Search-LWE. (In the event $\mathbf{A}$ is not invertible, call for more samples until it is.)

4.3 Solving for s

A variant of the previous strategy is to search for a suitable $\mathbf{s}$ directly such that $\|\mathbf{As} - \mathbf{c}\|$ is small. This literally solves Search-LWE. While this and the previous technique are related by simple linear algebra, i.e. knowing $\mathbf{e}$ trivially allows to recover $\mathbf{s}$ and vice versa, they differ in which of $\mathbf{e}$ or $\mathbf{s}$ they target.

5 Algorithms

5.1 Exhaustive Search

Exhaustive search directly solves for $\mathbf{s}$ as in Section 4.3.

Theorem 1. *The time complexity of solving* Search-LWE *with exhaustive search is $(\alpha q)^n \cdot 2n = 2^{n \log \alpha q + \log n + 1}$. The memory complexity is n. The sample complexity is $2n$.*

Proof. Apply Lemma 1 to obtain an LWE instance with $\mathbf{s}_i \leftarrow \mathcal{D}_{\mathbb{Z},\alpha q}$. We are therefore able to estimate the size of each component of the secret as $\mathrm{E}[\|\mathbf{s}_i\|] \leq \alpha q$. Therefore, to check all possible secrets we must enumerate approximately $(\alpha q)^n$ vectors. For each vector we perform about $2n$ operations in $\mathbb{Z}_q$ when computing the inner product.

Corollary 2. *Let $q = n^c$ and $\alpha q = \sqrt{n}$. Then the time complexity of solving* Search-LWE *with exhaustive search is $(\alpha q)^n \cdot 2n = 2^{\frac{1}{2}n \log n + \log n + 1}$. The memory complexity is n. The sample complexity is n.*

Remark 2. The complexity is independent of α and q but depends on their product αq and n.

Meet-in-the-Middle A Meet-in-the-Middle (MITM) attack also directly solves for $\mathbf{s}$ as in Section 4.3. This is a time-memory trade-off and hence a faster method than a naive brute force but at the cost of an increased requirement on memory.

Theorem 2. *Let the LWE instance be parametrised by n, α, q. Then the time taken to solve* Search-LWE *using MITM is $(\alpha q)^{n/2} \cdot 2n = 2^{\frac{1}{2}n \log \alpha q + \log n + 1}$. The memory complexity is $(\alpha q)^{n/2}$.*

Proof. Apply Lemma 1 to obtain an LWE instance with $\mathbf{s}_i \leftarrow \mathcal{D}_{\mathbb{Z},\alpha q}$. Given a sample $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$, split $\mathbf{a}$ in half and for each possibility of $\mathbf{s}$ compute the inner products of the first halves of $\mathbf{a}$ and $\mathbf{s}$ and the inner products of the second halves. Call the set of these inner products I_0 and I_1. Construct a table indexed by elements of I_0 and I_1, so that, for example, the inner product i in I_0 is listed with all choices of (the first half of) $\mathbf{s}$ such that $\langle (\mathbf{a}_{(0)}, \ldots, \mathbf{a}_{(n/2)-1}), (\mathbf{s}_{(0)}, \ldots, \mathbf{s}_{(n/2-1)}) \rangle = i$. We will need about $2n$ operations per entry for computing the inner product. Now search for near collisions i.e. $i_0 \in I_0$, $i_1 \in I_1$ with $i_0 + i_1$ small. Tracing back to a choice for the relevant halves of $\mathbf{s}$, we can guess $\mathbf{s}$ and be correct with non-negligible probability.

Corollary 3. *Let $q = n^c$ and $\alpha q = \sqrt{n}$. Then, the time complexity of solving* Search-LWE *using MITM is $(\alpha q)^n 2n = 2^{\frac{1}{4}n \log n + \log n + 1}$. The memory complexity is $(\alpha q)^{n/2} = 2^{\frac{n}{4} \log n}$.*

5.2 BKW

The BKW (Blum, Kalai, Wasserman) algorithm was introduced in [BKW03] and shows that subexponential algorithms exist for learning parity functions in the presence of noise. The BKW algorithm solves the LPN (Learning Parity with Noise) problem in time $2^{\mathcal{O}(n/\log n)}$. LPN can be considered as a special case of LWE with $q = 2$. BKW can be adapted to solve LWE [Reg09] and the complexity of this has been studied in [ACF+13]. In particular, BKW solves LWE via the SIS strategy (cf. Section 4.1).

To solve with this strategy, given m samples $(\mathbf{A}, \mathbf{c})$ following $L_{\mathbf{s},\chi}$, we require short vectors $\mathbf{v}_i$ in the scaled (by q) dual lattice of the lattice generated by the rows of $\mathbf{A}$. BKW constructs these by adding elements from a tables with q^b entries each, where each table is used to find collisions on b components of $\mathbf{a}$ (a row of $\mathbf{A}$).

In more detail, BKW constructs the $\mathbf{v}_i$ as follows. Given a sample $\mathbf{a}$, BKW splits the n components into a blocks each of width b. There are a stages of the algorithm in which the algorithm creates tables by searching for collisions in the appropriate b coefficients of $\mathbf{a}$. In the first stage after an appropriate number of samples we obtain two vectors which agree on $\mathbf{a}_{(0)}, \ldots, \mathbf{a}_{(b-1)}$. The algorithm will then take these and subtract them producing a row with $\mathbf{a}_{(0)} = \cdots = \mathbf{a}_{(b-1)} = 0$ which is stored for use in the next stage (considering $\mathbf{a}_{(b)}, \ldots, \mathbf{a}_{(2b-1)}$).

The $\mathbf{v}_i$ are of length $\sqrt{2^a}$. In the first stage, suppose we find a collision with the first b components. Adding those vectors clearing the first b components in $\mathbf{a}$ produces a $\mathbf{v}_i$ candidate of length $\sqrt{2}$ as we are adding two vectors. Moving on to the next stage, two such vectors are added to clear the next b columns, resulting in a $\mathbf{v}_i$ candidate of length $\sqrt{2^2}$, and so on for all a stages.

The algorithm maintains a tables of size q^b where $b = n/a$ and its running time is typically dominated by this magnitude. In general, we have the following complexity for solving Decision-LWE with BKW.

Theorem 3 ([ACF+13]). *Let $(\mathbf{a}_i, c_i)$ be samples following $L_{\mathbf{s},\chi}$ or a uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$, $0 < b \leq n$ be a parameter, $0 < \epsilon < 1$ the targeted success rate and $a = n/b$ the addition depth. Then, the expected cost of the BKW algorithm to distinguish $L_{\mathbf{s},\chi}$ from random with success probability ϵ is*

$$\left(\frac{q^b - 1}{2}\right) \cdot \left(\frac{a(a-1)}{2} \cdot (n+1) - \frac{ba(a-1)}{4} - \frac{b}{6}\left((a-1)^3 + \frac{3}{2}(a-1)^2 + \frac{1}{2}(a-1)\right)\right) \tag{2}$$

additions/subtractions in $\mathbb{Z}_q$ to produce elimination tables,

$$m \cdot \left(\frac{a}{2} \cdot (n+2)\right) \text{ with } m = \epsilon / \exp\left(-\pi\, \alpha^2\, 2^a\right) \tag{3}$$

additions/subtractions in $\mathbb{Z}_q$ to produce samples. Furthermore,

$$a \cdot \left\lceil \frac{q^b}{2} \right\rceil + m \tag{4}$$

calls to $L_{\mathbf{s},\chi}$ and storage for

$$\left(\frac{q^b}{2}\right) \cdot a \cdot \left(n + 1 - b\frac{a-1}{2}\right) \tag{5}$$

elements in $\mathbb{Z}_q$ are needed.

To pick a and b, recall from Remark 1 that in order to distinguish $L_{\mathbf{s},\chi}$ from random using SIS an appropriately short choice for $\mathbf{v}_i$ is $\|\mathbf{v}_i\| \cdot \alpha q \leq q$ hence a suitable choice for a is

$$\begin{aligned} \sqrt{2^a} \cdot \alpha q &\leq q \\ \sqrt{2^a} &\leq \alpha^{-1} \\ a &\leq \log(\alpha^{-2}). \end{aligned}$$

Corollary 4 ([ACF+13]). *Let $a = -2\log\alpha$ and $b = n/a$. The expected cost of the BKW algorithm to distinguish $L_{\mathbf{s},\chi}$ from random is*

$$\begin{aligned} \left(\frac{q^b - 1}{2}\right) \cdot \left(\frac{a(a-1)}{2} \cdot (n+1)\right) + \mathrm{poly}(n) &\leq q^b \cdot a^2 n + \mathrm{poly}(n) \\ &= \mathcal{O}\left(q^{n/(-2\log\alpha)} \cdot (-2\log\alpha)^2 n\right) \\ &= \mathcal{O}\left(2^{n\log q/(-2\log\alpha)} \cdot (-2\log\alpha)^2 n\right) \end{aligned}$$

operations in $\mathbb{Z}_q$ Furthermore, $a \cdot \left\lceil \frac{q^b}{2} \right\rceil + \mathrm{poly}(n)$ calls to $L_{\mathbf{s},\chi}$ and storage for

$$\left(\frac{q^b}{2}\right) \cdot a \cdot n$$

elements in $\mathbb{Z}_q$ are needed.

Specialising Corollary 4 with $q = n^c$ and $\alpha q = \sqrt{n}$ we get:

Corollary 5. *Let $q = n^c, \alpha\, q = \sqrt{n}$. Set $a = -2 \log \alpha$ and $b = n/a$. The expected cost of the BKW algorithm to distinguish $L_{\mathbf{s},\chi}$ from random is*

$$\begin{aligned}\left(\frac{q^b - 1}{2}\right) \cdot \left(\frac{a(a-1)}{2} \cdot (n+1)\right) + \text{poly}(n) &\leq q^b \cdot \left(a^2 n\right) + \text{poly}(n) \\ &= \mathcal{O}\left(2^{n \log q / -2 \log \alpha} \cdot \text{poly}(n)\right) \\ &= \mathcal{O}\left(2^{\frac{cn \log n}{(2c-1) \log n}} \cdot \text{poly}(n)\right) \\ &= \mathcal{O}\left(2^{\frac{n}{2-(1/c)}} \cdot \text{poly}(n)\right)\end{aligned}$$

operations in $\mathbb{Z}_q$.

Remark 3. It is easy to see that the complexity of the BKW algorithm is determined by n and αq and not α or q. However as q grows the leading coefficient of the complexity approaches $1/2$ as $1/c$ vanishes. This shows that in some sense the 'limit' of BKW is $\mathcal{O}\left(2^{(1/2)n}\right)$.

We note, however, that this strategy of picking a and b is not optimal. These choices, which produce an easy, closed form for the complexity, ensure that $m = \text{poly}(n)$, which implies that almost all time is spent constructing 'elimination tables', whereas the second step of the algorithm – producing candidates for distinguishing – is very efficient. A better strategy is to balance both steps, i.e. to find a and b such that

$$\left(\frac{q^b - 1}{2}\right) \cdot \left(\frac{a(a-1)}{2} \cdot (n+1)\right) = \epsilon / \exp\left(-\pi\, \alpha^2\, 2^a\right) \cdot \left(\frac{a}{2} \cdot (n+2)\right).$$

While this can make a significant difference for picking concrete parameters, it does not change the asymptotic behaviour of the algorithm and its performance characteristics.

Example 1. Choosing $n = 256$, $q = 65537$ and $\sigma = \alpha q / \sqrt{2\pi} \approx 25.53$ we expect the cost of distinguishing $L_{\mathbf{s},\chi}$ to be $2^{188.1}$ operations in $\mathbb{Z}_q$ by Corollary 4. Balancing the two steps of the algorithm would reduce this to 2^{171} operations in $\mathbb{Z}_q$.

Finally, we note that a reference implementation of the BKW algorithm for LWE is available as [Alb13].

5.3 Using Lattice Reduction To Distinguish

Lattice reduction is another means to find short vectors in the scaled dual lattice, enabling us to solve LWE via the SIS strategy. Again we consider the scaled dual lattice $L = \{\mathbf{w} \in \mathbb{Z}_q^m \mid \mathbf{w}\mathbf{A} \equiv 0 \mod q\}$. To construct this lattice from a given $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$: compute a basis $\mathbf{B}$ for the nullspace of $\mathbf{A}^T$ over $\mathbb{Z}_q$, lift to $\mathbb{Z}$ and extend by $q\mathbf{I} \in \mathbb{Z}^{m \times m}$ to make it q-ary and compute a basis for L. The lattice L has dimension m, and with high probability rank m and volume $\text{vol}(L) = q^n$ [MR09].

By our convention lattice reduction will return the shortest non-zero vector $\mathbf{b}_0$ it found as the first vector of a reduced basis, which by definition is a short vector in L, so that $\mathbf{b}_0\, \mathbf{A} = 0 \mod q$. Heuristically, for a good enough output basis all vectors could be used, as they will all be somewhat short, i.e. not too dissimilar in length from each other.

Lemma 7. *Let an LWE instance be parametrised by n, α, q. Any lattice reduction algorithm achieving log root-Hermite factor*

$$\log \delta_0 = \frac{\log^2\left(\alpha \frac{1}{f(\epsilon)}\right)}{4n \log q}$$

can distinguish $L_{\mathbf{s},\chi}$ with probability ϵ.

Proof. With high probability $\mathrm{vol}\,(L) = q^n$ and by definition the Hermite factor is $\delta_0^m = \frac{\|\mathbf{v}\|}{\mathrm{vol}(L)^{\frac{1}{m}}}$ so we have $\|\mathbf{v}\| = \delta_0^m\, q^{\frac{n}{m}}$. On the other hand, we require $\|\mathbf{v}\| = \frac{1}{\alpha} f(\epsilon)$ by Corollary 1. By Section 3.3 the optimal subdimension m which minimises the quantity $\delta_0^m q^{\frac{n}{m}}$ is $m = \sqrt{\frac{n \log q}{\log \delta_0}}$. Since we assume we can choose any number of samples m, we always choose to use this optimal subdimension. Rearranging with this value of m, we obtain $\log \delta_0 = \frac{\log^2\left(\frac{1}{\alpha} f(\epsilon)\right)}{4n \log q} = \frac{\log^2\left(\alpha \frac{1}{f(\epsilon)}\right)}{4n \log q}$ as our desired log root-Hermite factor. □

Corollary 6. *Given an LWE instance parametrised by n, $q = n^c$, $\alpha q = \sqrt{n}$. Any lattice reduction algorithm achieving log root-Hermite factor*

$$\log \delta_0 = \frac{\left(\left(c - \frac{1}{2}\right) \log n + \log f(\epsilon)\right)^2}{4cn \log n}$$

can distinguish $L_{\mathbf{s},\chi}$ with probability ϵ.

Proof.

$$\delta_0^m q^{\frac{n}{m}} = \|\mathbf{v}\|$$

$$\delta_0^m n^{\frac{cn}{m}} = n^{c-\frac{1}{2}} f(\epsilon)$$

$$\sqrt{\frac{cn \log n}{\log \delta_0}} \log \delta_0 + \frac{cn}{\sqrt{\frac{cn \log n}{\log \delta_0}}} \log n = \left(c - \frac{1}{2}\right) \log n + \log f(\epsilon)$$

$$\frac{cn \log n \log \delta}{\log \delta_0} + cn \log n = \sqrt{\frac{cn \log n}{\log \delta_0}} \left(\left(c - \frac{1}{2}\right) \log n + \log f(\epsilon)\right)$$

$$2cn \log n = \sqrt{\frac{cn \log n}{\log \delta_0}} \left(\left(c - \frac{1}{2}\right) \log n + \log f(\epsilon)\right)$$

$$\frac{2cn \log n}{\left(\left(c - \frac{1}{2}\right) \log n + \log f(\epsilon)\right)} = \sqrt{\frac{cn \log n}{\log \delta_0}}$$

$$\frac{(2cn \log n)^2}{\left(\left(c - \frac{1}{2}\right) \log n + \log f(\epsilon)\right)^2} = \frac{cn \log n}{\log \delta_0}$$

$$\frac{4cn \log n}{\left(\left(c - \frac{1}{2}\right) \log n + \log f(\epsilon)\right)^2} = \frac{1}{\log \delta_0}$$

$$\frac{\left(\left(c - \frac{1}{2}\right) \log n + \log f(\epsilon)\right)^2}{4cn \log n} = \log \delta_0$$

□

Remark 4. Assuming $q = n^c$ and $\alpha q = \sqrt{n}$ we can see that for large q and hence large c, lattice reduction becomes easier, as we get a larger δ_0. Contrasting this with BKW, we can see while that algorithm is somewhat competitive in time complexity with lattice reduction for small q, it is much worse than the latter for large q as they are, for example, used in homomorphic encryption schemes [GHS12b] (cf. Section 7).

Having established the target δ_0, we can combine it with estimates about lattice reduction running times from Section 3.2. In Table 2 we list estimates for how long it would take lattice reduction algorithms to achieve our target δ_0 for $f(\epsilon) = 1$, i.e. $\epsilon \approx 1/23$.

model	block size k	log clock cycles
rule of thumb	$\frac{k}{\log k} = \frac{4n \log q}{\log^2\left(\frac{1}{\alpha}\right)}$	$\mathcal{O}(k)$
simp. rule of thumb	$\frac{4n \log q}{\log^2\left(\frac{1}{\alpha}\right)}$	$\mathcal{O}\left(\frac{4n \log q}{\log^2\left(\frac{1}{\alpha}\right)}\right)$
Lindner & Peikert	?	$\frac{7.2n \log q}{\log^2\left(\frac{1}{\alpha}\right)} - 78.9$
delta-squared model	?	$\frac{0.144n^2 \log^2 q}{\log^4\left(\frac{1}{\alpha}\right)} + 4.1$
$q = n^c, \alpha = n^{1/2-c}$		
rule of thumb	$\frac{k}{\log k} = \frac{4cn \log n}{\left(\left(c-\frac{1}{2}\right) \log n\right)^2}$	$\mathcal{O}(k)$
simp. rule of thumb	$\frac{4cn \log n}{\left(\left(c-\frac{1}{2}\right) \log n\right)^2}$	$\mathcal{O}\left(\frac{4cn \log n}{\left(\left(c-\frac{1}{2}\right) \log n\right)^2}\right)$
Lindner & Peikert	?	$\frac{7.2cn \log n}{\left(\left(c-\frac{1}{2}\right) \log n\right)^2} - 78.9$
delta-squared model	?	$\frac{0.144c^2n^2 \log^2 n}{\left(\left(c-\frac{1}{2}\right) \log n\right)^4} + 4.1$

Table 2. Time complexity for distinguishing $L_{\mathbf{s},\chi}$ from random based on lattice reduction estimates from the literature.

Considering the right-most column of Table 2 it is clear that both the Lindner-Peikert model as well as the simplified lattice rule of thumb would predict a subexponential running time for solving LWE with SIS. Since this is widely assumed to not be the case, we may discount these approximations as too optimistic.

As pointed out in Section 4.1 above, the strategy as discussed so far is not optimal. Given access to sufficiently many samples m it is usually beneficial to run lattice reduction for a smaller target success probability ϵ' and to repeat this process $\approx \epsilon/\epsilon'$ times to boost the overall success probability to the desired success probability.

Example 2. Setting $n = 256$, $q = 65537$ and $\sigma = \alpha q/\sqrt{2\pi} \approx 25.53$ following [Reg09] and picking $\epsilon = 0.1$ we get a target $\delta_0 = 1.004051$ by Lemma 7 and thus $m = 838$. Computing the expected number of clock cycles according to the various models available to us, we end up with the following estimates.

model	block size k	log clock cycles
rule of thumb	832	≈ 832
simp. rule of thumb	172	≈ 172
Lindner & Peikert	?	223
delta-squared model	?	269
fplll	391	1990
bkz2	391	436
sieve	391	185

In contrast, picking ϵ' such that ϵ/ϵ' multiplied by the solving time is minimised we get:

model	$\log(\epsilon')$	δ_0	block size k	log clock cycles
rule of thumb	-66	1.006042	585	≈ 585
simp. rule of thumb	-15	1.005005	154	≈ 154
Lindner & Peikert	-29	1.005454	?	179
delta-squared model	-35	1.005586	?	178
fplll	-51	1.005854	224	700
bkz2	-28	1.005430	252	207
sieve	-21	1.005232	267	170

5.4 Decoding Attack

This attack solves LWE via the BDD strategy (cf. [LP11]). The most basic way of solving a BDD instance is using Babai's Nearest Plane algorithm [Bab85]. This attack can be summarised as follows: let there be m samples of an LWE instance parametrised by n, α, q so we have a set of samples $(\mathbf{A}, \mathbf{c})$. Perform lattice reduction on the lattice $L(\mathbf{A}^T)$ to obtain a new basis $\mathbf{B}$ for this lattice, where the quality of this basis is characterised as usual by the root-Hermite factor δ_0. Babai's Nearest Plane algorithm works by recursively computing the closest vector on the sublattice spanned by subsets of the Gram-Schmidt vectors $\mathbf{b}_i^*$.

This recovers the vector $\mathbf{s}$ with probability

$$\prod_{i=0}^{m-1} \operatorname{erf}\left(\frac{\|\mathbf{b}_i^*\|\sqrt{\pi}}{2\alpha q}\right)$$

under the assumption that sampling from the discrete Gaussian is approximately the same as sampling from a continuous Gaussian [LP11].

The probability the nearest planes algorithm finds the vector $\mathbf{s}$ is given by the probability that the error vector $\mathbf{e}$ lies in the parallelepiped $\mathbf{s} + \mathcal{P}(\mathbf{B}^*)$. So, it can be seen that in this approach the success probability is determined by the quality of the lattice reduction.

Lindner and Peikert Nearest Planes Lindner and Peikert [LP11] suggest an alteration of Babai's algorithm, designed to widen the fundamental parallelepiped in the direction of $\mathbf{b}_i^*$ by a factor of some $d_i \in \mathbb{Z}_{>0}$, thereby increasing the chance of $\mathbf{e}$ falling inside it. This will find multiple solutions, which can be searched through exhaustively to find the correct solution.

This modifies the success probability to

$$\prod_{i=0}^{m-1} \operatorname{erf}\left(\frac{d_i \cdot \|\mathbf{b}_i^*\|\sqrt{\pi}}{2\alpha q}\right). \tag{6}$$

There is no obvious way to analytically determine the optimal d_i to achieve a desired success probability. However, Lindner and Peikert suggest a simple heuristic method in which d_i are chosen to maximise $\min_{1 \le i \le m} (d_i \cdot \|\mathbf{b}_i^*\|)$. This can be shown to return optimal values if we restrict our d_i to powers of 2 only. Since $\frac{\operatorname{erf}(2x)}{\operatorname{erf}(x)} > \frac{\operatorname{erf}(2y)}{\operatorname{erf}(y)}$ for all $0 < x < y$, then clearly the optimal value is obtained by doubling d_i whenever $d_i \cdot \|\mathbf{b}_i\|$ is minimal. Therefore, maximising the minimum of the values $d_i \cdot \|\mathbf{b}_i\|$ is optimal for d_i powers of 2.

Given m, n, α and q as above, let $t_{NP}(\delta_0, \epsilon) = t_{node} \cdot \prod_{i=0}^{m-1} d_i$ such that Equation (6) is at least ϵ where t_{node} is the number of clock cycles it takes to visit one node. Then the time for a decoding attack to achieve a success probability ϵ could be determined as

$$t_{dec}(\epsilon) = \min_{\delta_0} \{t_{BKZ}(\delta_0) + t_{NP}(\delta_0, \epsilon)\}.$$

Hence, on the one hand, with a more reduced a basis, the values of d_i can be smaller, so the Nearest Planes algorithm requires less time. On the other hand, the lattice reduction takes significantly more time for smaller approximation factors.

We note that in [LP11, Figure 4] it appears as though this quantity has not been optimised. The authors find values for δ_0 for which the time of a decoding attack is less than an equivalent distinguishing attack (cf. Section 5.3), but these values are not necessarily optimal, i.e. the lattice reduction step and the decoding step are not always balanced.

We note that just as in Section 5.3 we may opt to run the attack many times with a lower advantage. This typically improves the overall complexity.

Solving BDD by Enumeration: an Update (Liu, Nguyen) Liu and Nguyen [LN13] note that the Lindner Peikert algorithm (as well as Babai's) can be viewed as a form of pruned enumeration, but with a different rule to Gama, Nguyen and Regev's pruned enumeration [GNR10]. Namely, let $\mathbf{v}$ be a node and $\mathbf{t}$ be a target vector. GNR pruning keeps nodes with bounded projections whereas the Lindner Peikert algorithm keeps nodes with bounded coordinates, in particular $|\zeta_i(\mathbf{v} - \mathbf{t})| \leq d_i||\mathbf{b}_i^*||/2$ where $\zeta_i(\mathbf{x}) = \frac{\langle \mathbf{x}, \mathbf{b}_i^* \rangle}{||\mathbf{b}_i^*||}$. Liu and Nguyen note that this can be generalised to arbitrary bounds on coordinates, $|\zeta_i(\mathbf{v} - \mathbf{t})| \leq R_i$ for some parameters R_i not necessarily dependent on the $||\mathbf{b}_i^*||$s.

Due to these similarities between the Lindner Peikert method and pruning techniques, Liu and Nguyen implement a variant of the LP algorithm in the context of pruning algorithms, using arbitrary R_i. They also randomise the input basis, allowing them to repeat the algorithm multiple times, which has the result of increasing both the runtime and success probability linearly. Since we assume access to as many samples as required, we do not rely on rerandomisation when estimating complexity. These two factors result in more flexibility in tuning the parameters, and improved results for solving BDD.

However, instead of using the enumeration framework as simply a method to improve the algorithm of Lindner and Peikert, Liu and Nguyen go on to directly apply pruned enumeration to solve BDD. This follows the earlier work of Gama, Nguyen and Regev [GNR10], and uses linear pruning in which the bounds $R_k = \sqrt{k/m} R_m$ are used. Over the same parameters used in [LP11], this linear pruning is shown to improve on both the original Nearest Planes algorithm and the improved variant.

Runtime Analysis In any lattice decoding attack, the runtime is determined by balancing the lattice reduction step against the final step which enumerates possible solutions and outputs an answer with a certain probability.

For Babai's algorithm, the runtime is determined by calculating the Gram-Schmidt orthogonalisation - which can be done with floating point arithmetic in $\mathcal{O}(n^3)$. If using either of the extensions to Babai's algorithm this is still a component, but the main factor determining the runtime is the number of points which are calculated.

Similarly, if using a form of enumeration, we are mostly interested in how many points are enumerated. Therefore, to calculate the runtime of the BDD attack, we simplify the various enumeration

algorithms to two expressions: the time it takes to enumerate one point; and the success probability for a certain number of enumerations.

For example, Lindner and Peikert estimate that running Babai's algorithm once takes $t_{node} = 2^{-16} \cdot 2.33 \cdot 10^9 \approx 2^{15.1}$ clock cycles, whereas [GNR10] achieve $0.94 \cdot 10^7$ nodes per second which is approximately 2^{-23} seconds per enumeration. In our estimator (cf. Section 7) we assume $t_{node} = 2^{15.1}$.

Calculating the success probability is harder. For Nearest Planes, we can use Equation 6, but we still need to determine the optimal values for d_i for which we do not know a closed formula. In practice, though, we can follow Lindner & Peikert's strategy of increasing d_i one by one. For enumeration and pruning, we need to use the method as used in [LN13], which experimentally calculates the success probability by sampling.

The most significant factor affecting the success probability is the quality of the reduced basis which is provided (i.e. what value δ_0 is achieved). For example, Babai's algorithm without a preceding lattice reduction only gives solutions up to an exponential factor. In a sense, the methods proposed here for performing a decoding attack can be seen as a way to halt the lattice reduction when it is possible to obtain a solution with a reasonable success probability, and optionally repeating to increase the probability of solving the problem. The overall runtime is then calculated by estimating the optimal time to halt the reduction and attempt to solve.

Example 3. For $n = 192, q = 4093, \alpha q = 8.87$ [LP11] report 2^{74} seconds when running the attack 2^{32} times with advantage 2^{-32} and $\delta_0 = 1.0083$ whereas the randomised NP used by Liu and Nguyen is able to perform the decoding attack using a lattice reduction with $\delta_0 = 1.0077$ and $\epsilon = 2^{-12}$. This lattice reduction takes $2^{65.6}$ seconds in the Lindner & Peikert model for lattice reduction. Our estimator suggests $\epsilon = 2^{-9}$ and $\delta_0 = 1.0076$ which implies a lattice reduction cost of $2^{63.8}$ seconds also in the Lindner & Peikert model (for compatibility).

We note that these improvements depend on balancing many parameters in an optimal way. Calculating the success probability can only be done numerically, and optimising parameters requires many computations. Our estimator (cf. Section 7) does not provide a routine for estimating the cost using [LN13] but we restrict out attention to [LP11] which gives comparable results and is easier to estimate.

5.5 Reducing BDD to uSVP

Albrecht, Fitzpatrick and Göpfert [AFG13] consider the complexity of solving LWE via BDD by reducing BDD to uSVP (unique Shortest Vector Problem). It is folklore that solving BDD via uSVP is generally the most efficient strategy. Formally, the γ-uSVP problem is as follows: given a lattice L such that $\lambda_2(L) > \gamma\lambda_1(L)$, find a shortest nonzero vector in L.

To reduce BDD to uSVP Kannan's embedding technique [Kan87] is used. The idea is to embed $L(\mathbf{A}) = \{\mathbf{Au} \mid \mathbf{u} \in \mathbb{Z}_q^n\}$, the lattice generated by the columns of the LWE instance (and our usual lattice for consideration when are solving with the BDD strategy), into a higher-dimensional lattice $L(\mathbf{B})$ with γ-uSVP structure. That is, $\mathbf{B}$ is constructed as

$$\mathbf{B} = \begin{pmatrix} \tilde{\mathbf{A}} & 0 \\ \mathbf{c} & t \end{pmatrix},$$

where $\tilde{\mathbf{A}}$ is a basis for the q-ary lattice spanned by the columns of $\mathbf{A}$.

Let $\mathbf{y} \in L$, for some lattice L, be the closest lattice point to some point $\mathbf{x}$, i.e. the point minimising $\|\mathbf{x} - \mathbf{y}\|$. We can then define the distance from $\mathbf{x}$ to the lattice L, $\text{dist}(\mathbf{x}, L)$, as this length. If the embedding factor $t = \text{dist}(\mathbf{c}, L(\mathbf{A})) < \frac{\lambda_1(L(\mathbf{A}))}{2\gamma}$ then $\text{L}(\mathbf{B})$ contains a γ-unique shortest vector, $\mathbf{c}' = (\mathbf{e}, -t)$ [LM09], which if we can find, we can take the first m components to recover $\mathbf{e}$, hence solving also the BDD instance.

To solve a γ-uSVP instance, we may reduce the problem to κ-HSVP (Hermite Shortest Vector Problem). Let $\gamma = \kappa^2$. Lovasz [Lov86] showed that any algorithm which can solve κ-HSVP, such as a lattice reduction algorithm, can be used linearly many times to solve approximate SVP with approximation factor κ^2. Intuitively, a lattice with uSVP structure has one direction in which its shortest vector is somewhat shorter than all other directions. A sufficiently precise lattice reduction algorithm (for example) can produce a vector so short it must be in this special direction. More precisely, a solution to κ^2-approximate SVP would be a vector $\mathbf{v}$ such that $\|\mathbf{v}\| \leq \kappa^2 \lambda_1(L)$. On the other hand, any vector $\mathbf{w}$ which is not the shortest (and independent of the shortest vector) satisfies $\|\mathbf{w}\| \geq \lambda_2(L) > \kappa^2 \lambda_1(L)$. So, we must have $\mathbf{v}$ is a multiple of a shortest vector, and hence we have solved κ^2-uSVP. Luzzi et al. [LLLS11] show that whenever $\kappa > \sqrt{N}$, for N the dimension of the lattice, this result can be improved. They show any algorithm solving κ-HSVP can be used to solve γ-uSVP, where $\gamma \approx \sqrt{N}\kappa$.

The above are theoretical results. In practice, an algorithm solving HSVP will solve uSVP instances where the gap is $\lambda_2(L) > \tau \delta_0^m \lambda_1(L)$ with some probability depending on τ. The value τ is taken to be a constant, which is experimentally derived in [GN08] and which depends on both the nature of the lattices considered, the lattice reduction algorithm used and the target success rate.

To estimate the time complexity of this approach we firstly must establish the variables m and τ. The determination of τ is discussed at length in [AFG13] but essentially we have that $\tau \approx 0.3$ is a fair estimate for success probability 10% based on the experiments mentioned above as well as in [AFG13]. We stress that no data is publicly available on τ for smaller success probability. Determining m depends on how we choose the embedding factor t. We may have $t = \|\mathbf{e}\|$ or $t < \|\mathbf{e}\|$.

Suppose firstly that $t = \|\mathbf{e}\|$. We will need the following lemma from [AFG13].[1]

Lemma 8 (Lemma 2 in [AFG13]). *Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, let $\alpha q > 0$ and let $\epsilon' > 1$. Let $\mathbf{e}$ be drawn from $\mathcal{D}_{\mathbb{Z}^m, \alpha q}$. Under the assumption that $\lambda_1(L(\mathbf{A})) \geq \sqrt{\frac{m}{2\pi e}} \text{vol}\,(L)^{1/m}$ and that the rows of $\mathbf{A}$ are linearly independent over $\mathbb{Z}_q$, we can create an embedding lattice with λ_2/λ_1-gap greater than*

$$\frac{\min\{q, \frac{q^{1-\frac{n}{m}} \Gamma(1+\frac{m}{2})^{\frac{1}{m}}}{\sqrt{\pi}}\}}{\frac{\epsilon' s \sqrt{m}}{\sqrt{\pi}}} \approx \frac{\min\{q, q^{1-\frac{n}{m}} \sqrt{\frac{m}{2\pi e}}\}}{\frac{\epsilon' s \sqrt{m}}{\sqrt{\pi}}}$$

with probability greater than $1 - (\epsilon' \cdot \exp(1 - \epsilon'^2)/2)^m$.

Hence, setting $t = \|\mathbf{e}\|$, $\mathbf{B}$ is a basis of a lattice whose gap is determined by Lemma 8. Using Lemma 8 and under the assumption $q^{1-\frac{n}{m}} \sqrt{\frac{m}{2\pi e}} < q$, we require a gap of approximately size

$$\frac{\lambda_2}{\lambda_1} = \frac{q^{1-\frac{n}{m}} \sqrt{\frac{1}{2e}}}{\epsilon' \alpha q}$$

[1] To avoid notational conflict we refer to their constant $c > 1$ as a constant $\epsilon' > 1$ and we replace their s for the width parameter of the Gaussian with our αq.

and so we require a δ_0 determined by $q^{1-\frac{n}{m}}\sqrt{\frac{1}{2e}} \geq \tau\delta_0^m\,\epsilon'\alpha q$. In [AFG13] it is shown that for a fixed δ_0 the optimal subdimension is $m = \sqrt{\frac{n\log q}{\log \delta_0}}$ as in Section 3.3. We may use this to determine δ_0 using the expression above (where for simplicity we assume equality).

Lemma 9. *Given an LWE instance characterised by n, α, q. Any lattice reduction algorithm achieving log root-Hermite factor*

$$\log \delta_0 = \frac{\log^2\left(\tau\epsilon'\alpha\sqrt{2e}\right)}{4n\log q}$$

solves LWE, by reducing BDD to uSVP, with success probability greater than

$$\epsilon_\tau \cdot \left(1 - (\epsilon' \cdot \exp\left(1-\epsilon'^2\right)/2)^m\right)$$

for some fixed $\tau \leq 1$ and $0 < \epsilon_\tau < 1$ as a function of τ.

Proof. From the above discussion, and assuming for simplicity an equality, we require a δ_0 determined by the following equation:

$$q^{\left(1-\frac{n}{\sqrt{\frac{n\log q}{\log \delta_0}}}\right)}\sqrt{\frac{1}{2e}} = \tau\,\epsilon'\,\alpha q\,\delta_0^{\sqrt{\frac{n\log q}{\log \delta_0}}}$$

Rearranging, we obtain

$$\begin{aligned}
\left(1 - \frac{n}{\sqrt{\frac{n\log q}{\log \delta_0}}}\right)\log q + \log\sqrt{\frac{1}{2e}} &= \log\left(\tau\epsilon'\alpha q\right) + \sqrt{\frac{n\log q}{\log \delta_0}}\log\delta_0 \\
\left(\frac{\sqrt{\frac{n\log q}{\log \delta_0}} - n}{\sqrt{\frac{n\log q}{\log \delta_0}}}\right)\log q - \sqrt{\frac{n\log q}{\log \delta_0}}\log\delta_0 &= \log\left(\tau\epsilon'\alpha q\right) - \log\sqrt{\frac{1}{2e}} \\
\left(\frac{\sqrt{\frac{n\log q}{\log \delta_0}} - n}{\sqrt{\frac{n\log q}{\log \delta_0}}}\right)\log q - \sqrt{\frac{n\log q}{\log \delta_0}}\log\delta_0 &= \log\left(\tau\epsilon'\alpha q\sqrt{2e}\right) \\
\sqrt{\frac{n\log q}{\log \delta_0}}\log\left(\tau\epsilon'\alpha\sqrt{2e}\right) &= 2n\log q \\
\frac{n\log q}{\log \delta_0}\log^2\left(\tau\epsilon'\alpha\sqrt{2e}\right) &= 4n^2(\log q)^2 \\
\frac{\log^2\left(\tau\epsilon'\alpha\sqrt{2e}\right)}{4n\log q} &= \log\delta_0.
\end{aligned}$$

Finally, the success probability is computed as the probability the gap is as required in Lemma 8 multiplied by the success probability of our algorithm ϵ_τ. □

Corollary 7. *Given an LWE instance characterised by n, $q = n^c$, $\alpha q = \sqrt{n}$. Any lattice reduction algorithm achieving log root-Hermite factor*

$$\log \delta_0 = \frac{\left((c-1/2)\log n - \log\left(\tau\epsilon'\sqrt{2e}\right)\right)^2}{4cn\log n}$$

solves LWE, by reducing BDD to uSVP, with success probability greater than

$$\epsilon_\tau \cdot \left(1 - (\epsilon' \cdot \exp\left(1-\epsilon'^2\right)/2)^m\right)$$

for some fixed $\tau \leq 1$ *and* $0 < \epsilon_\tau < 1$ *in function of* τ.

Secondly, suppose $t < \|\mathbf{e}\|$. In this case no efficient method for determining $\frac{\lambda_2}{\lambda_1}$ is known. The assumption in [AFG13] which attempts to overcome this is that the same size of gap is required as it is in the case that $t = \|\mathbf{e}\|$. A modified value for τ is then derived under this assumption. Setting $t = 1$ is typically more efficient than $t = \|\mathbf{e}\|$, see [AFG13] for details.

Comparing Corollary 7 with Corollary 6 we find that solving LWE via BDD by reducing to uSVP is more efficient than solving LWE via one call to an algorithm solving SIS whenever $\log(1/(\tau\epsilon'\sqrt{2e})) > \log(f(\epsilon))$ under the condition that $\epsilon = \epsilon_\tau \cdot \left(1 - (\epsilon' \cdot \exp\left(1-\epsilon'^2\right)/2)^m\right)$ so that the success probabilities are equal in both cases.

Example 4. Letting $n = 256$, $q = 65537$ and $\sigma = \alpha q/\sqrt{2\pi} \approx 25.53$ following [Reg09] and choosing $t = 1$, $\tau = 0.310$ and $\epsilon' = 1.01$ we get a target $\delta_0 = 1.004634$ by Lemma 9 and thus $m = 783$. According to [AFG13] we have $\epsilon_\tau = 0.1$. Computing the expected number of clock cycles according to the various models available to us, we end up with the following estimates.

model	block size k	log clock cycles
rule of thumb	710	≈ 710
simp. rule of thumb	150	≈ 150
Lindner & Peikert	?	191
delta-squared model	?	206
fplll	321	1344
bkz2	321	307
sieve	321	165

The time complexity reported here is smaller than in the first example of Example 2 but not necessarily smaller than the data reported in the second example there. Since the only values for τ reported in [AFG13] correspond to $\epsilon_\tau = 0.1$, we cannot estimate how BDD-via-uSVP would behave in the low advantage regime.

5.6 Arora-Ge and Gröbner Bases

Arora and Ge proposed an alternative approach to solving Search-LWE by setting up a system of noise free non-linear polynomials of which the secret $\mathbf{s}$ is a root [AG11]. This approach solves for $\mathbf{s}$ directly.

In particular, [AG11] offers an algorithm for solving Search-LWE in time $2^{\tilde{\mathcal{O}}(n^{2\xi})}$, where ξ is a constant such that $\alpha q = n^\xi$. The algorithm proceeds by assuming that the error always falls in the range $[-t, t]$ for some $t \in \mathbb{Z}$ such that $d = 2t + 1 < q$. This follows from the chance of falling outside this interval dropping exponentially fast. In particular, we have the following standard fact about the Gaussian distribution:

Lemma 10. *Let χ denote the Gaussian distribution with standard deviation σ. Furthermore, for $x > 0$, we denote $Q(x) = \frac{1}{2}\left(1 - \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$. Then, for all $C > 0$, it holds that:*

$$\Pr[e \leftarrow \chi : |e| > C \cdot \sigma] \approx 2 \times Q(C) \leq \frac{2}{C\sqrt{2\pi}} e^{-C^2/2} \in e^{\mathcal{O}(-C^2)}.$$

From this fact, polynomials are constructed from the observation that the error, when falling in this range, is always a root of the polynomial $P(x) = x\prod_{i=1}^{d}(x+i)(x-i)$. Then, we know the secret $\mathbf{s}$ is a root of $P(\mathbf{a}\cdot\mathbf{x} - c)$ constructed from LWE samples. In Arora-Ge the system of non-linear equations constructed this way is solved by linearisation. However, this means that we need $\mathcal{O}\left(n^{2t+1}\right)$ samples. As we increase the number of samples, we increase the probability that the error falls outside of the interval $[-t, t]$. We then have to increase the range, leading to a larger degree, which requires even more samples. Balancing these two requirements of keeping the degree low and acquiring enough samples, the overall complexity is given by the following result.

Theorem 4 (Theorem 5 in [ACF+14]). *Let $n, q, \sigma = \alpha q$ be parameters of an LWE instance, and as before let ω denote the linear algebra constant. Let $D_{\mathrm{AG}} = 8\sigma^2 \log n + 1$. If $D_{\mathrm{AG}} \in o(n)$ then the Arora-Ge algorithm solves* Search-LWE *in time complexity*

$$\mathcal{O}\left(2^{\omega \cdot D_{\mathrm{AG}} \log \frac{n}{D_{\mathrm{AG}}}} \cdot \sigma q \log q\right) = \mathcal{O}\left(2^{8\,\omega\,\sigma^2 \log n(\log n - \log(8\,\sigma^2 \log n))} \cdot \mathrm{poly}(n)\right)$$

and memory complexity

$$\mathcal{O}\left(2^{2 \cdot D_{\mathrm{AG}} \log \frac{n}{D_{\mathrm{AG}}}} \cdot \sigma q \log q\right) = \mathcal{O}\left(2^{16\,\sigma^2 \log n(\log n - \log(8\,\sigma^2 \log n))} \cdot \mathrm{poly}(n)\right).$$

If $n \in o(D_{\mathrm{AG}})$ then the Arora-Ge algorithm solves Search-LWE *in time complexity*

$$\mathcal{O}\left(2^{\omega \cdot n \log \frac{D_{\mathrm{AG}}}{n}} \cdot \sigma q \log q\right) = \mathcal{O}\left(2^{\omega\, n \log(8\,\sigma^2 \log n) - n \log n} \cdot \mathrm{poly}(n)\right)$$

and memory complexity

$$\mathcal{O}\left(2^{2n \log \frac{D_{\mathrm{AG}}}{n}} \cdot \sigma q \log q\right) = \mathcal{O}\left(2^{2n \log(8\,\sigma^2 \log n) - n \log n} \cdot \mathrm{poly}(n)\right).$$

This can be improved by using Gröbner basis techniques [ACF+14]. In particular, to solve via linearisation as in [AG11], we require $\mathcal{O}\left(n^d\right)$ equations, but Gröbner basis algorithms will work when fewer equations than this are available at the cost of a more expensive solving step. In particular, the complexity of computing a Gröbner basis is $\mathcal{O}\left(\binom{n+D_{\mathrm{reg}}}{D_{\mathrm{reg}}}^{\omega}\right)$, where D_{reg} is the degree of regularity of the ideal $\mathcal{I}$ spanned by the polynomials. The degree D_{reg} is the index of the first non-positive coefficient of the Hilbert series expansion of the ideal $\mathcal{I}$. In general, it is hard to compute the Hilbert series, but for semi-regular sequences, it has an easy form. The Hilbert series of a semi-regular sequence with m polynomials of degree d in n variables is:

$$H_{\mathcal{I}}(z) := \frac{(1-z^d)^m}{(1-z)^n}.$$

It is assumed that random systems behave like semi-regular sequences. Thus, assuming our non-linear equations behave like random equations of the same degree, we can estimate the cost of solving LWE by expanding this power-series until the first non-positive coefficient. In particular, assuming $\alpha q = \sqrt{n}$ we get:

Theorem 5 ([ACF+14]). *Let* $(\mathbf{a}_i, b_i)$ *for* $i \geq 1$ *be elements of* $\mathbb{Z}_q^n \times \mathbb{Z}_q$ *sampled according to* $L_{\mathbf{s},\chi}$ *with* $\alpha q = \sqrt{n}$. *There is an algorithm recovering the secret with time complexity* $\mathcal{O}\left(2^{2.82\omega n}\right)$, *memory complexity* $\mathcal{O}\left(2^{5.64n}\right)$ *and sample complexity* $m = \exp(\frac{\pi}{4} \cdot n)$.

Remark 5. The complexity solely depends on αq, which corresponds to the degree, and n, which corresponds to the number of variables. Adjusting q while keeping αq the same will not affect the runtime.

6 Small Secret Variants

In several applications based on LWE, the secret $\mathbf{s}$ is not chosen uniformly at random from $\mathbb{Z}_q$ but instead chosen from a different distribution where all the components $\mathbf{s}_{(i)}$ are "small", e.g. they are chosen from $\{0, 1\}$ or $\{-1, 0, 1\}$. In this section we consider the complexity of solving LWE in this special case. We characterise an instance by n, α, q, ψ where ψ is the distribution of $\mathbf{s}_{(i)}$. We assume we have as many samples m as we need.

6.1 Exhaustive Search

In Section 5.1 above we saw that exhaustive search can be solved by checking all the vectors within a sphere of radius αq which is essentially the size of the secret. Even without explicitly knowing ψ, we can restrict our search to the support of ψ, for example $\{-1, 0, 1\}$. We can simply check all possible $\mathbf{s}$ with $\mathbf{s}_{(i)}$ chosen from this set. Then by the same argument as in Theorem 1, exhaustive search will take time $3^n \cdot (2n) = 2^{n \log 3 + \log n + 1}$ if $\mathbf{s}_{(i)} \in \{-1, 0, 1\}$.

MITM By exactly the same argument as in Theorem 2, whatever time we would expect it to take to solve exhaustive search (which depends on ψ), we may achieve essentially the same speed up as we would do applying a Meet-in-the-Middle strategy to a general LWE instance. So, if the components $\mathbf{s}_{(i)}$ are selected from $\{-1, 0, 1\}$ then an MITM strategy will take time $3^{n/2} \cdot (2n) = 2^{\frac{1}{2} n \log 3 + \log n + 1}$ and require $3^{n/2}$ memory.

6.2 Lattice Reduction for Small Secret LWE

For an LWE instance parametrised by n, α, q and with a small secret, we may apply modulus switching and consider the instance mod p where $p < q$. This allows for a larger δ_0 than would be required for an instance parametrised by the same n, α, q and with a secret where $\mathbf{s}_{(i)}$ is chosen from all $\mathbb{Z}_q$. After modulus switching, the transformed instance has error which is slightly larger and its distribution is no longer exactly a discrete Gaussian. Nonetheless, heuristically, algorithms which solve LWE still solve these LWE-like problem instances and so we assume that after modulus switching, we have an LWE instance characterised by n, $\sqrt{2}\alpha$ and p. So, for any algorithm with a lattice reduction step (for example, those described in Sections 5.3, 5.4 and 5.5), in the case where we have a small secret, we may obtain a speed up by modulus switching before performing the lattice reduction, and then continuing to run the algorithms as described above.

As an example we consider distinguishing LWE by lattice reduction as in Section 5.3. As with a general secret, we assume the size of the small vector we aim to output is $\|\mathbf{v}\| = \frac{1}{\alpha} f(\epsilon)$.

Lemma 11. *Let a small secret LWE instance be characterised by n, α, q and $\mathbf{s}_{(i)} \leftarrow \psi$. Then the log root-Hermite factor $\log \delta_0$ required to distinguish by lattice reduction is*

$$\log \delta_0 = \frac{\left(\log\left(\sqrt{2}\alpha \frac{1}{f(\epsilon)}\right)\right)^2}{4n \log p}$$

for p such that

$$\left\| \left\langle \frac{p}{q} \cdot \mathbf{a} - \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle \right\| \approx \frac{p}{q} \cdot \|e\| .$$

Proof. Using Lemma 2, modulus switch and transform the LWE instance $(\mathbf{a}, c) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ into an LWE instance in $\mathbb{Z}_p^n \times \mathbb{Z}_p$. Now the instance is parametrised by $n, p, \sqrt{2}\alpha$ and by the same argument as in Lemma 7 we require $\delta_0 = 2^{\frac{\left(\log\left(\sqrt{2}\alpha \frac{1}{f(\epsilon)}\right)\right)^2}{4n \log p}}$. □

Corollary 8. *Let a small secret LWE instance be characterised by n, α, q and ψ, suppose $\alpha q = \sqrt{n}$ and $q = n^c$ and ψ is such that the standard deviation of the elements in the secret $\mathbf{s}$ is σ_s. Then the log root-Hermite factor $\log \delta_0$ that is required is*

$$\log \delta_0 = \frac{\left(\log\left(\frac{\sqrt{2}}{f(\epsilon)} n^{\frac{1}{2}-c}\right)\right)^2}{4n\left(\log\left(\frac{\sqrt{\pi}}{\sqrt{6}}\sigma_s\right) + c \log n\right)}$$

Proof. From Lemma 2 we have $p = \frac{\sigma_s}{\alpha}\sqrt{\frac{2\pi n}{12}} = \frac{\sigma_s \sqrt{2}\sqrt{\pi}\sqrt{n}}{\sqrt{12}\alpha} = \frac{\sqrt{\pi}}{\sqrt{6}}\sigma_s n^c$.
By Lemma 11 we have

$$\log \delta_0 = \frac{\left(\log\left(\sqrt{2}\alpha \frac{1}{f(\epsilon)}\right)\right)^2}{4n \log p}$$

$$= \frac{\left(\log\left(\sqrt{2}\alpha \frac{1}{f(\epsilon)}\right)\right)^2}{4n \log\left(\frac{\sqrt{\pi}}{\sqrt{6}}\sigma_s n^c\right)}$$

$$= \frac{\left(\log\left(\frac{\sqrt{2}}{f(\epsilon)} n^{\frac{1}{2}-c}\right)\right)^2}{4n\left(\log\left(\frac{\sqrt{\pi}}{\sqrt{6}}\sigma_s\right) + c \log n\right)}$$

□

Example 5. Setting $n = 256$, $q = 65537$ and $\sigma = \alpha q/\sqrt{2\pi} \approx 25.53$ following [Reg09] and $\psi = \mathcal{U}(\mathbb{Z}_2)$ we have $\sigma_s = 0.5$. Modulus switching reduces the size of the modulus to $p = 5928$. Picking $\epsilon = 0.1$ we get a target $\delta_0 = 1.00465678$ by Lemma 7 and thus $m = 781$. Computing the expected number of clock cycles according to the various models available to us, we end up with the following estimates.

model	block size k	log clock cycles
rule of thumb	706	≈ 706
simp. rule of thumb	150	≈ 150
Lindner & Peikert	?	190
delta-squared model	?	204
fplll	318	1312
bkz2	318	295
sieve	318	157

6.3 BKW Small Secret

In this section we consider [AFFP14]. In this work ψ is not specified but it is assumed that the $\mathbf{s}_{(i)}$ are chosen from $\{-1, 0, 1\}$ or $\{0, 1\}$. The authors employ their own variant of BKW to achieve a complexity reduction for solving BKW with small secret. Their technique is *lazy modulus switching*, a variant of modulus switching. To maximise complexity improvements, the authors only modulus switch when necessary, and employ techniques such as searching for collisions mod p but remaining in $\mathbb{Z}_q$ when doing arithmetic on the rows.

Theorem 6 ([AFFP14]). *Let $n \geq 1$ be the dimension of the LWE secret vector, q be a modulus, $b \in \mathbb{Z}$ with $1 \leq b \leq n$ and σ_s the standard deviation of the secret vector components. Let also σ_r be the variance of random elements in $\mathbb{Z}_r$. Define $a = \lceil n/b \rceil$ and pick a pair (p, m^*) such that $b\,\sigma_r^2\,\sigma_s^2 \sum_{i=0}^{a-1} \mathbf{v}_{(i)} \leq 2^a \sigma$. Then $B_{\mathbf{s},\chi}(b, a-1, p)$ will return $(\tilde{\mathbf{a}}_0, \tilde{c}_0), \ldots, (\tilde{\mathbf{a}}_{m-1}, \tilde{c}_{m-1})$ where $\tilde{c}_i$ has standard deviation $\leq \sqrt{2^{a+1}}\sigma$. Furthermore this costs $\frac{p^b}{2}\left(\frac{a(a-1)}{2}(n+1)\right) + (m+m^*)na$ additions in $\mathbb{Z}_q$ and $\frac{ap^b}{2} + m + m^*$ calls to $L_{\mathbf{s},\chi}$.*

In particular, for a typical choice of parameters: $q \approx n^c$ for some small $c \geq 1$, $a = \log n$, $b = \frac{n}{\log n}$, recall that standard BKW has complexity $\mathcal{O}\left(2^{cn} \cdot n \log^2 n\right)$. Here, using naive modulus switching, the complexity of solving is $\mathcal{O}\left(2^{n(c+\frac{\log d}{\log n})} \cdot n \log^2 n\right)$. Using lazy modulus switching (Corollary 3 of [AFFP14]), the complexity of solving is $\mathcal{O}\left(2^{n(c+\frac{\log d - \frac{1}{2}\log\log n}{\log n})} \cdot n \log^2 n\right)$, where in both cases $0 < d \leq 1$ is a constant.

6.4 Arora-Ge and Gröbner Bases

We may exploit small secrets when reducing LWE to solving a non-linear system of equations as in Section 5.6. To encode that our secret is small, we add low-degree equations of the form $\prod_{i=0}^{s-1} x - j_i$ where s is the cardinality of the support for ψ and j_i are the elements of the support. We may then expand the Hilbert series to establish the expected degree of semi-regularity.

7 Examples

In this section we use our estimator to apply techniques discussed in Sections 5 and 6 to parameter sets from the literature. Our estimator is available at `https://bitbucket.org/malb/lwe-estimator`. We consider the following parameter sets.

Regev	These are Regev's example choices for parameters from [Reg09]. We use [AFC+13] to pick $q \approx n^2$ and $\alpha = 1/(\sqrt{2\pi n}\log_2^2 n)$.
LindnerPeikert	We use [AFC+13] to select parameters as suggested in [LP11] given n.
FHE	Given n and the multiplicative depth L we set $q = 2^{16.5 \cdot L + 5.4} \cdot 8^{2L-3} \cdot n^L$ and $\alpha = \sqrt{2\pi} \cdot 3.2/q$ inspired by parameters suggested in [GHS12c]. We always assume $\mathbf{s}_{(i)} \leftarrow \{0,1\}$.

In our tables "MITM" refers to the Meet-in-the-Middle algorithm given in Section 5.1, "BKW" to the BKW algorithm discussed in Section 5.2, "SIS" to the algorithm discussed in Section 5.3, "BDD" to the algorithm discussed in Section 5.4, "uSVP" to the algorithm discussed in Section 5.5 and "Arora-GB" to applying Gröbner basis algorithms as discussed in Section 5.6. In those tables concerning small secret variants, the same labels refer to the small secret variants of the respective algorithms. The columns "ops" refer to estimated bit operations which we identify with CPU clock cycles. This identification slightly favours lattice reduction algorithms compared to other algorithms, because CPUs do more than one operation per bit per clock cycle. The columns "mem" refer to storage requirements of elements in $\mathbb{Z}_q$. The columns "calls" refers to the number of calls to the LWE oracle. All columns list the logarithm to base two of their respective values. The columns "renum" resp. "rsieve" refer to BKZ 2.0 estimates based on the row "enum" resp. "sieve" in Table 1. We use the "renum" estimates to optimise parameters for "SIS" and "BDD". The column "enum" gives the number of enumerations in the decoding stage of "BDD". If "–" is listed instead of a number, it means our estimator did not return a value or was not run because it does not cover this particular case. This can happen when estimates only exist for special cases such as when applying Gröbner bases.

	MITM			BKW			SIS			BDD			uSVP		
n	ops	mem	calls	ops	mem	calls	renum	rsieve	calls	ops	enum	calls	renum	rsieve	calls
64	133.2	129.6	7.0	55.2	48.1	43.0	39.9	51.5	9.7	39.5	23.6	8.5	46.9	58.5	10.7
128	324.6	320.8	8.0	98.3	90.7	84.6	66.1	83.4	20.5	59.7	43.6	15.5	58.3	76.0	11.8
256	781.0	777.0	9.0	182.8	173.3	167.4	207.7	170.5	60.4	180.3	164.5	36.4	225.1	146.1	12.9
512	1847.2	1843.0	10.0	354.9	346.6	338.5	932.3	335.0	103.5	642.6	626.8	85.4	1371.3	308.2	13.9
1024	4293.7	4289.4	11.0	697.7	689.2	680.1	4591.7	678.5	181.5	2185.6	2169.7	232.3	7983.9	678.4	15.0

Table 3. Regev

	MITM			BKW			SIS			BDD			uSVP		
n	ops	mem	calls	ops	mem	calls	renum	rsieve	calls	ops	enum	calls	renum	rsieve	calls
64	42.6	32.0	7.0	42.2	35.4	30.3	39.3	53.0	8.5	39.0	23.2	8.2	46.2	57.8	10.5
128	75.8	64.0	8.0	72.1	64.8	58.7	56.8	74.6	17.3	51.0	35.0	11.3	51.7	67.4	11.5
256	141.0	128.0	9.0	130.5	122.9	115.9	153.7	138.5	40.2	135.4	119.5	28.2	152.5	124.4	12.6
512	270.2	256.0	10.0	239.9	229.7	223.3	607.5	280.9	93.1	456.1	440.7	73.1	851.7	251.7	13.6
1024	527.3	512.0	11.0	456.7	448.6	439.5	2845.6	548.9	154.1	1514.1	1498.6	184.0	4809.4	537.3	14.6

Table 4. Regev with $\mathbf{s}_{(i)} \leftarrow \{0,1\}$

	MITM			BKW			SIS			BDD			uSVP			Arora-GB		
n	ops	mem	calls	ops	mem	calls	renum	rsieve	calls	ops	enum	calls	renum	rsieve	calls	ops	mem	calls
64	104.4	100.9	7.0	52.3	45.2	40.2	39.6	51.1	9.5	39.1	23.2	8.4	46.6	58.1	10.6	289.3	285.9	92.1
128	188.2	184.8	8.0	85.8	78.7	72.6	61.2	79.0	20.3	55.6	40.0	13.3	54.4	71.6	11.6	504.8	501.3	389.3
256	405.4	401.8	9.0	153.5	146.1	139.1	171.6	154.5	55.2	151.3	135.7	34.2	180.4	133.2	12.6	–	–	–
512	746.2	742.6	10.0	277.7	266.3	261.6	681.3	303.0	106.1	495.5	479.7	85.0	1051.7	274.6	13.6	–	–	–
1024	1624.9	1621.2	11.0	528.4	520.7	511.6	3037.9	588.4	183.0	1561.9	1545.9	181.9	5734.1	581.8	14.6	–	–	–

Table 5. LindnerPeikert

	MITM			BKW			SIS			BDD			uSVP			Arora-GB		
n	ops	mem	calls	ops	mem	calls	renum	rsieve	calls	ops	enum	calls	renum	rsieve	calls	ops	mem	calls
64	42.5	32.0	7.0	42.1	35.3	30.2	39.3	50.8	9.4	39.0	23.2	8.2	46.2	57.8	10.5	249.6	246.3	257.2
128	75.5	64.0	8.0	70.2	63.5	57.4	64.5	82.2	22.2	58.4	42.6	13.3	56.3	73.9	11.5	504.7	501.3	534.4
256	140.6	128.0	9.0	128.5	121.4	114.4	188.9	164.5	61.2	165.2	149.3	36.2	207.6	140.9	12.6	–	–	–
512	269.6	256.0	10.0	240.2	233.2	225.1	854.0	339.1	120.1	588.6	572.9	103.0	1433.8	313.3	13.6	–	–	–
1024	526.7	512.0	11.0	471.1	463.8	454.8	3876.5	664.5	209.1	1841.2	1825.9	211.9	7868.4	673.0	14.7	–	–	–

Table 6. LindnerPeikert with $\mathbf{s}_{(i)} \leftarrow \{0, 1\}$

	MITM			BKW			SIS			BDD			uSVP		
n	ops	mem	calls	ops	mem	calls	renum	rsieve	calls	ops	enum	calls	renum	rsieve	calls
64	44.7	32.0	7.0	52.9	41.0	36.0	37.1	48.7	8.1	37.8	21.8	8.0	45.8	57.4	10.3
128	77.8	64.0	8.0	84.3	72.4	66.4	40.3	51.8	9.1	41.1	25.1	9.0	49.0	60.6	11.4
256	142.8	128.0	9.0	146.4	134.4	127.4	43.5	55.0	10.1	44.4	28.5	10.0	52.2	63.7	12.4
512	271.9	256.0	10.0	269.7	257.6	249.6	46.6	58.1	11.1	47.6	31.7	11.1	55.3	66.8	13.4
1024	528.9	512.0	11.0	515.2	503.0	493.9	67.7	84.3	13.1	64.6	48.5	12.1	70.3	87.9	14.4
2048	1042.0	1024.0	12.0	1006.3	994.0	984.0	283.1	168.5	25.1	257.9	242.1	19.1	273.6	163.9	15.4

Table 7. FHE with $L = 2$ and $\mathbf{s}_{(i)} \leftarrow \{0, 1\}$

	MITM			BKW			SIS			‘	‘BDD		uSVP		
n	ops	mem	calls	ops	mem	calls	renum	rsieve	calls	ops	enum	calls	renum	rsieve	calls
64	47.1	32.0	7.0	61.4	44.7	39.7	36.9	48.4	8.0	37.9	21.9	8.0	45.8	57.3	10.3
128	80.2	64.0	8.0	94.1	77.4	71.3	40.0	51.6	9.0	41.3	25.4	9.0	48.9	60.4	11.3
256	145.2	128.0	9.0	158.5	141.6	134.6	43.2	54.7	10.0	44.2	28.2	10.0	52.0	63.6	12.3
512	274.3	256.0	10.0	286.2	269.2	261.2	46.3	57.8	11.0	47.9	32.2	11.0	55.2	66.7	13.3
1024	531.3	512.0	11.0	539.6	522.5	513.5	49.4	60.9	12.0	52.1	36.8	12.0	58.3	69.8	14.3
2048	1044.4	1024.0	12.0	–	–	–	52.5	64.0	13.0	56.9	41.7	13.0	61.3	72.9	15.3
4096	2069.4	2048.0	13.0	–	–	–	55.6	67.1	14.0	58.7	43.4	14.0	64.4	76.0	16.3
8192	4118.5	4096.0	14.0	–	–	–	108.7	111.1	15.0	111.0	95.8	15.0	115.2	118.9	17.3
16384	8215.5	8192.0	15.0	–	–	–	565.1	220.3	19.0	549.0	533.8	16.0	561.3	224.7	18.3

Table 8. FHE with $L = 10$ and $\mathbf{s}_{(i)} \leftarrow \{0, 1\}$

8 Discussion

The problems of giving the concrete hardness of the Learning with Errors problem are manifold.

No closed formulas. For most algorithms, there is no closed formula which expresses the running time in terms of the parameters specifying the problem (e.g. n, q, α). This makes direct comparisons difficult. This problem is addressed by the Sage module, enabling us to estimate running times of the various algorithms for particular parameter choices.

The results of applying this Sage module broadly agree with the literature. By our estimates the parameter choices made in [GHS12c] are too conservative, as first observed by van de Pol and Smart [vdPS13]. This is because their parameters were chosen assuming Lindner and Peikert's estimate for the runtime of BKZ, which we rule out from among the choices of estimates because it implies a subexponential algorithm for solving LWE.

No single best algorithm. Our results indicate that there is not one algorithm which always outperforms all others on the parameter sets we tested, and so we cannot recommend to consider one particular algorithm to achieve security level λ. Which algorithm performs best depends on the concrete parameters considered. For small n, BDD may be favourable (see e.g. Table 5). For large n, BKW may be fastest when considering public-key encryption (see e.g. Table 3) but not when considering homomorphic encryption schemes which require large q. For the particular case of the FHE parameters we considered (Table 7 and Table 8) we see that BKW is far slower than the other algorithms, for a given n.

Moreover, we cannot even rule out certain algorithms from consideration. For example, while the Arora-Ge algorithm and its Gröbner basis variants always perform much worse than other algorithms in our tests, it is shown in [ACF+14] that this family of algorithms outperforms other families when considering a particular variant of LWE, i.e. UniformNoise-LWE instances. Similarly, as expected, the MITM approach performs considerably worse than other approaches in the uniform secret setting, but is rather competitive in the small secret setting.

Time-memory trade-offs. According to our estimates of running BKZ, for larger n, using sieving as the SVP oracle is faster than BKZ 2.0 as practically implemented using enumeration. This is to be expected given that sieving is asymptotically faster than enumeration. It is important to note, however, that sieving would require an amount of memory so substantial that taking both time and memory into consideration, for most parameters we consider, both require about the same amount of total resources. Hence, taking both memory and time into consideration it is not clear that sieving is worth considering for reasonable security levels, such as, say, $\lambda = 128$. A completely analogous statement can be made of comparing an exhaustive search with a Meet-in-the-Middle attack or when considering the BKW algorithm.

Incomplete data. Our estimator results appear to show that for any n, and for whichever SVP oracle, BDD is faster than uSVP. This at first sight appears to contradict the folklore that a reduction to uSVP is the fastest way to solve LWE. However, a reason for this discrepancy is that we are able to optimise parameters for BDD but we do not know how to do this for uSVP. As highlighted in Section 5.5, for success probability 10% we have that $\tau \approx 0.3$ is a fair estimate based on experiments in the literature, but no data is publicly available from which to estimate τ for smaller success probabilities.

This is just one area in which more data is required. Our estimator is built from the curves fitted to the data from the literature given in Table 1 and as such more experimental data on the

runtime of enumeration and sieving would be very valuable in terms of refining the estimates. To reiterate, the analysis on which the estimator is based is sound given the current state of the art, but intrinsically depends on the formulae for sieving and enumeration, and so refinements in this area will refine our estimator accordingly. As lattice reduction is a central step in many of the algorithms, this is of particular importance.

Another possible direction of future work is developing algorithms for small secret LWE. As mentioned above, MITM performs fairly competitively with other algorithms in this case, but intuitively this is about the worst possible algorithm - one would expect that essentially an exhaustive search be somewhat slower than a specially designed algorithm for the problem. This may suggest there is more to be done in this area.

Acknowledgements. Albrecht was supported by EPSRC grant EP/L018543/1 "Multilinear Maps in Cryptography". Player was supported by an ACE-CSR PhD grant. Scott was supported by EPSRC grant EP/K035584/1.

References

ACF+13. Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Designs, Codes and Cryptography*, pages 1–30, 2013.

ACF+14. Martin Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Algebraic algorithms for LWE problems. Cryptology ePrint Archive, Report 2014/1018, 2014. `http://eprint.iacr.org/2014/1018`.

ACPS09. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Halevi [Hal09], pages 595–618.

ACPS14. Martin Albrecht, David Cadé, Xavier Pujol, and Damien Stehlé. fplll, development version. Available at `https://github.com/dstehle/fplll`, 2014.

AFC+13. Martin R. Albrecht, Robert Fitzpatrick, Daniel Cabracas, Florian Göpfert, and Michael Schneider. A generator for LWE and Ring-LWE instances, 2013. available at `http://www.iacr.org/news/files/2013-04-29lwe-generator.pdf`.

AFFP14. Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Lazy modulus switching for the BKW algorithm on LWE. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 429–445. Springer, March 2014.

AFG13. Martin R. Albrecht, Robert Fitzpatrick, and Florian Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In Hyang-Sook Lee and Dong-Guk Han, editors, *ICISC 13*, volume 8565 of *LNCS*, pages 293–310. Springer, November 2013.

AG11. Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part I*, volume 6755 of *LNCS*, pages 403–415. Springer, July 2011.

Ajt96. Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.

Alb13. Martin Albrecht. BKW-LWE, 2013. `https://bitbucket.org/malb/bkw-lwe/`.

Bab85. László Babai. On lovász' lattice reduction and the nearest lattice point problem (shortened version). In Kurt Mehlhorn, editor, *STACS '86*, volume 82 of *Lecture Notes in Computer Science*, pages 13–20. Springer, 1985.

BBD09. Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors. *Post-Quantum Cryptography*. Springer, 2009.

BGJ13. Anja Becker, Nicolas Gama, and Antoine Joux. Solving shortest and closest vector problems: The decomposition approach. Cryptology ePrint Archive, Report 2013/685, 2013. `http://eprint.iacr.org/2013/685`.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

BKW03. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, July 2003.

BLP+13. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

Che13. Yuanmi Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Paris 7, 2013.

CHKP12. David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of Cryptology*, 25(4):601–639, October 2012.

CN11. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Lee and Wang [LW11], pages 1–20.

CPS13. David Cadé, Xavier Pujol, and Damien Stehlé. *fpLLL 4.0.4*, 2013. `http://perso.ens-lyon.fr/damien.stehle/fplll/`.

DB13. Léo Ducas-Binda. *Signatures Fondées sur les Réseaux Euclidiens: Attaques, Analyses et Optimisations*. PhD thesis, École Normale Supérieure Paris, 2013. `http://cseweb.ucsd.edu/~lducas/Thesis/index.html`.

Gen09. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

GGH13. Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, May 2013.

GGH14. Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. Cryptology ePrint Archive, Report 2014/645, 2014. `http://eprint.iacr.org/2014/645`.

GHS12a. Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482. Springer, April 2012.

GHS12b. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, August 2012.

GHS12c. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. *IACR Cryptology ePrint Archive*, 2012:99, 2012.

Gil10. Henri Gilbert, editor. *EUROCRYPT 2010*, volume 6110 of *LNCS*. Springer, May 2010.

GKPV10. Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In Andrew Chi-Chih Yao, editor, *ICS*, pages 230–240. Tsinghua University Press, 2010.

GN08. Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 31–51. Springer, April 2008.

GNR10. Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Gilbert [Gil10], pages 257–278.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

Hal09. Shai Halevi, editor. *CRYPTO 2009*, volume 5677 of *LNCS*. Springer, August 2009.

HJP14. William Hart, Fredrik Johansson, and Sebastian Pancratz. FLINT: Fast Library for Number Theory, 2014. Version 2.4.4, `http://flintlib.org`.

HPS11a. Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 447–464. Springer, August 2011.

HPS11b. Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Coding and Cryptology*, volume 6639 of *Lecture Notes in Computer Science*, pages 159–190. Springer, 2011.

Jou09. Antoine Joux. *Algorithmic Cryptanalysis*. Chapman & Hall/CRC, 1st edition, 2009.

Kan87. Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, August 1987.

LLL82. A.K. Lenstra, Jr. Lenstra, H.W., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

LLLS11. Cong Ling, Shuiyin Liu, Laura Luzzi, and Damien Stehlé. Decoding by embedding: Correct decoding radius and DMT optimality. *CoRR*, abs/1102.2936, 2011.

LM09. Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In Halevi [Hal09], pages 577–594.

LN13. Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 293–309. Springer, February / March 2013.

Lov86. László Lovász. *An algorithmic theory of numbers, graphs and convexity*. CBMS-NSF regional conference series in applied mathematics. Philadelphia, Pa. Society for Industrial and Applied Mathematics, 1986.

LP11. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, February 2011.

LvdPdW12. Thijs Laarhoven, Joop van de Pol, and Benne de Weger. Solving hard lattice problems and the security of lattice-based cryptosystems. Cryptology ePrint Archive, Report 2012/533, 2012. `http://eprint.iacr.org/2012/533`.

LW11. Dong Hoon Lee and Xiaoyun Wang, editors. *ASIACRYPT 2011*, volume 7073 of *LNCS*. Springer, December 2011.

MR09. Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Bernstein et al. [BBD09], pages 147–191.

MW14. Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. Cryptology ePrint Archive, Report 2014/569, 2014. `http://eprint.iacr.org/2014/569`.

Ngu10. Phong Q. Nguyen. Hermite's constant and lattice algorithms. In Phong Q. Nguyen and Brigitte Vallée, editors, *The LLL Algorithm*, Information Security and Cryptography, pages 19–69. Springer, 2010.

Ngu11. Phong Q. Nguyen. Lattice reduction algorithms: Theory and practice (invited talk). In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 2–6. Springer, May 2011.

NS05. Phong Q. Nguyen and Damien Stehlé. Floating-point LLL revisited. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 215–233. Springer, May 2005.

Pei09. Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009.

PVW08. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, August 2008.

PW11. Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. *SIAM J. Comput.*, 40(6):1803–1844, 2011.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):34:1–34:40, September 2009.

Reg10. Oded Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204. IEEE Computer Society, 2010.

S$^+$14. William Stein et al. *Sage Mathematics Software Version 6.3*. The Sage Development Team, 2014. `http://www.sagemath.org`.

Sho. V. Shoup. Number theory library 5.5.2 (ntl) for c++. `http://www.shoup.net/ntl/`.

Ste13. Damien Stehlé. An overview of lattice reduction algorithms. Invited talk at ICISC, 2013.

vDGHV10. Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Gilbert [Gil10], pages 24–43.

vdPS13. Joop van de Pol and Nigel P. Smart. Estimating key sizes for high dimensional lattice-based systems. In Martijn Stam, editor, *14th IMA International Conference on Cryptography and Coding*, volume 8308 of *LNCS*, pages 290–303. Springer, December 2013.