Review Article

Martin R. Albrecht, Rachel Player and Sam Scott

On the concrete hardness of Learning with Errors

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 use 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 LWE problem.

Keywords: Learning with Errors, lattice-based cryptography, lattice reduction

MSC 2010: 94A60, 11T71

Martin R. Albrecht, Rachel Player, Sam Scott: Information Security Group, Royal Holloway, University of London, UK, e-mail: martin.albrecht@rhul.ac.uk, rachel.player.2013@live.rhul.ac.uk, sam.scott.2012@live.rhul.ac.uk

Communicated by: Spyros Magliveras

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 [2] 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 [15]. A third reason is the wealth of applications of lattice-based cryptography, perhaps the most notable of which is its role in the realisation of fully homomorphic encryption by Gentry [32] and in follow up works (e.g., [17, 34, 78]). Lattice problems have also been the basis of Public Key Encryption [69], including CCA secure schemes [65]; Identity Based Encryption [37] and the more general Hierarchical Identity Based Encryption [22]; oblivious transfer schemes [66]; Circular-Secure Encryption [11]; and Leakage-Resilient Encryption [39]. Recently, candidate constructions for multi-linear maps based on presumed hard problems in lattices have been proposed [31, 33].

Learning with errors. One lattice problem on whose hardness several cryptographic constructions are based is the *learning with errors* (LWE) problem [65, 67, 69]. LWE was introduced by Regev in [69] and is provably as hard as worst-case lattice problems [18, 69]. It is a generalisation of the *learning parity with noise* (LPN) problem [47] into large moduli q.

Definition 1.1 (LWE [70]). 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 considering it in $\mathbb{Z}_q$, 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 [12] 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 tight closed formulae are known expressing their complexities, the third contribution of this survey is that we provide a module for the Sage mathematics software [76] 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 on $\mathbb{Z}$ 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 $\exp(-\pi(x-\mu)^2/(\alpha q)^2)$. The standard deviation of a continuous Gaussian with width parameter αq is $\sigma = \alpha q/\sqrt{2\pi}$ and we roughly have this relation when we discretise, as long as σ is bigger than the smoothing parameter $\eta_\varepsilon(\mathbb{Z})$ of $\mathbb{Z}$ (see [28]). For ease of analysis, some works (e.g., [52]) 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 in the following way.

(i) 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. Having $\alpha q > \sqrt{n}$ allows the reduction of GapSVP to LWE to go through [70]. 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 [12]. 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).

(ii) The most generic characterisation is by n, α, q.

(iii) In some applications, the components of the secret $\mathbf{s}$ are not chosen uniformly at random from $\mathbb{Z}_q$ but instead we have the guarantee that they are all "small", e.g., $\mathbf{s}_{(i)} \in \{0, 1\}$. 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 [71]. This could be explained by the result that given a fixed (polynomial) number of samples, one can generate arbitrarily many more (independent) samples, with only a slight worsening in the error [11, 37].

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 used later. In Section 4 we give 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 and 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. The concatenation of two vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted $(\mathbf{a} \parallel \mathbf{b})$. 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 any constant such that there is an algorithm which multiplies matrices in $\mathcal{O}(n^\omega)$ operations for sufficiently large n. Hence, our ω differs slightly from the definition in [20] which hides logarithmic factors (cf. [68]). Unless stated otherwise, we use $\approx$ to denote something sufficiently close to equal that we treat it as equal for our estimates.

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 $\lambda_1(L) \approx \sqrt{m/(2\pi e)}\,\mathrm{vol}(L)^{1/m}$. 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.

We now give four 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 2.1 ([11]). *Let $\mathcal{D}_{\mathbb{Z}^n,\alpha q}$ be an n-dimensional extension of $\mathcal{D}_{\mathbb{Z},\alpha q}$ to $\mathbb{Z}^n$ in the obvious way, i.e. 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 $\mathbf{a} \leftarrow_\$ \mathcal{U}(\mathbb{Z}_q^n)$, $e \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\alpha q}$ 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 $\mathbf{a} \leftarrow_\$ \mathcal{U}(\mathbb{Z}_q^n)$, $e \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\alpha q}$ and $\mathbf{e} \leftarrow_\$ \mathcal{D}_{\mathbb{Z}^n,\alpha q}$ in $2n^2$ operations in $\mathbb{Z}_q$ per sample, at the loss of n samples overall and with $\mathcal{O}(n^\omega)$ 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 $\mathbf{a} \leftarrow_\$ \mathcal{U}(\mathbb{Z}_q^n)$, $e \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\alpha q}$ 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 $\mathbf{a} \leftarrow_\$ \mathcal{U}(\mathbb{Z}_q^n)$, $e \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\alpha q}$ and $\mathbf{e} \leftarrow_\$ \mathcal{D}_{\mathbb{Z}^n,\alpha q}$. Finally, computing $\mathbf{a}_1 \cdot \mathbf{A}_0^{-1}$ takes $2n^2$ operations in $\mathbb{Z}_q$. □

Given samples from $L_{\mathbf{s},\chi}$, we may be able to construct LWE instances where the modulus is now p for some particular $p < q$ by *modulus switching*. This technique was initially introduced to speed-up homomorphic encryption [19] but can also be employed to reduce the cost of solving LWE in certain cases [8]. 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, which in general leads to a higher complexity for solving LWE. Below, we choose p to ensure that $\|\langle \frac{p}{q} \cdot \mathbf{a} - \lfloor \frac{p}{q} \cdot \mathbf{a} \rceil, \mathbf{s}\rangle\| \approx \frac{p}{q} \cdot \|e\|$ if $\mathbf{s}$ is small enough. This means that the new error term after modulus switching is essentially the previous error scaled. In particular, the new distribution is LWE with errors having standard deviation $\sqrt{2}\alpha p/\sqrt{2\pi} + O(1)$. Following [18, 19] we have the following lemma.

Lemma 2.2. *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 $\mathbf{s}$. Assume that the distribution of the secret $\mathbf{s}$ is such that it has mean 0. If $p < q$ then*

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

is sample from an LWE distribution with error of standard deviation $\sqrt{2}\alpha p/\sqrt{2\pi} + \mathcal{O}(1)$.

Proof. Consider

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

where e' is distributed in $(-0.5, 0.5]$ and e'' is an inner product of small n-dimensional vectors and thus approaches a discrete Gaussian as n tends to infinity. Its standard deviation is $\sqrt{n/12}\,\sigma_s$, since $\frac{p}{q} \cdot \mathbf{a} - \lfloor \frac{p}{q} \cdot \mathbf{a} \rceil$ takes values $\in (-0.5, 0.5]$. We have that $\frac{p}{q} \cdot e$ is a scaled discrete Gaussian of standard deviation

$$\frac{p}{q} \cdot \frac{\alpha q}{\sqrt{2\pi}} = \frac{\alpha p}{\sqrt{2\pi}}.$$

Targeting $e'' \approx \frac{p}{q} \cdot e$, i.e. that the standard deviations of both distributions are the same, we get

$$\frac{\alpha p}{\sqrt{2\pi}} \approx \sqrt{\frac{n}{12}}\,\sigma_s, \quad p \approx \sqrt{\frac{2\pi n}{12}} \cdot \frac{\sigma_s}{\alpha}.$$

Since the standard deviations of $\frac{p}{q} \cdot e$ and e'' are the same for this p, the standard deviation of the new distribution is $\sqrt{2}\alpha p/\sqrt{2\pi} + \mathcal{O}(1)$ as claimed. □

Following the literature, we assume that the new distribution output by the process described in Lemma 2.2 is $L_{\mathbf{s},\mathcal{D}_{\mathbb{Z},\sqrt{2}\alpha p+1}}$, i.e. that the error is discrete Gaussian, even for relatively small n.

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 non-trivial direction is due to Regev [70], 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 2.3 ([70, Lemma 4.2]). *Let $n \geq 1$ be some integer, $2 \leq q \leq \mathrm{poly}(n)$ 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 $k = \mathbf{s}_{(0)}$. □

In what follows we will also make use of the following standard fact about the Gaussian distribution.

Lemma 2.4. *Let χ denote the Gaussian distribution with standard deviation σ and mean zero. Then, for all $C > 0$, it holds that*

$$\Pr[e \leftarrow_{\$} \chi : |e| > C \cdot \sigma] \leq \frac{2}{C\sqrt{2\pi}} \exp(-C^2/2).$$

Proof. For $t > C \cdot \sigma$, we have $t/(C \cdot \sigma) > 1$. Hence, we have

$$\begin{aligned}
\Pr[e \leftarrow_{\$} \chi : |e| > C \cdot \sigma] &= 2 \cdot \int_{C\cdot\sigma}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{t^2}{2\sigma^2}\right) dt \\
&= \frac{2}{\sqrt{2\pi}} \int_{C\cdot\sigma}^{\infty} \frac{1}{\sigma} \exp\left(-\frac{t^2}{2\sigma^2}\right) dt \\
&\leq \frac{2}{\sqrt{2\pi}} \int_{C\cdot\sigma}^{\infty} \frac{t}{C\,\sigma^2} \exp\left(-\frac{t^2}{2\sigma^2}\right) dt \\
&= \frac{2}{C\,\sqrt{2\pi}} \exp(-C^2/2)\, dt.
\end{aligned}$$

□

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 [23, 52, 58, 61, 62] for further details on lattices.

Lattice reduction algorithms can be viewed as a hierarchy: cases of BKZ [73] based on the block size parameter k. For $k = 2$ the algorithm runs in polynomial time but the reduced basis output will only be LLL-reduced, i.e. it will only contain a short vector to within exponential factors of a shortest vector. 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 at least 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 [45]). The output of the Lagrange/Gauss 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 $\{\mathbf{b}_0, \mathbf{b}_1\}$ arranged such that $\|\mathbf{b}_0\| \leq \|\mathbf{b}_1\|$ 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., [64] 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 $\delta \in (\frac{1}{4}, 1)$. 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 [50]. In particular, for an input basis where for all i, $\|\mathbf{b}_i\| < B$, LLL outputs an LLL-reduced basis in time $\mathcal{O}(n^{5+\varepsilon} \log^{2+\varepsilon} B)$ (using fast integer multiplication). In more recent variants, improvements have been made. For example, one variant introduced by Nguyen and Stehlé called L2 [63] provably outputs an LLL-reduced basis in time $\mathcal{O}(n^{5+\varepsilon} \log B + n^{4+\varepsilon} \log^2 B)$ (using fast integer multiplication). That is, one that only grows quadratically in $\log B$. Heuristically, variants of LLL achieve $\mathcal{O}(n^3 \log^2 B)$ (see [24]).

Quality of output. LLL theoretically achieves a Hermite factor of $(\frac{4}{3})^{\frac{n-1}{4}}$ (see[50]). In practice, it behaves much better and a root-Hermite factor δ_0 of 1.0219 is reported in [29].

Implementations. LLL and its variants are implemented in many software packages, notably in NTL [74], FLINT [43] and fplll [21]. The latter also implements L2.

3.2 BKZ

The BKZ algorithm [73] 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 [41]. 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.

An HKZ reduced basis $\{\mathbf{b}_0, \ldots, \mathbf{b}_{n-1}\}$ is a basis such that its Gram–Schmidt vectors $\mathbf{b}_i^*$ satisfy $\|\mathbf{b}_i^*\| = \lambda_1(\pi_i(L))$ for $0 \leq i \leq n-1$ 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 an 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 [24]. These improvements are extreme pruning [30], early termination, limiting the enumeration radius to the Gaussian heuristic and local block pre-processing. Extreme pruning takes place in the enumeration subroutine, and it works by exploring only certain branches in the search tree, with the hope that a short enough vector is still found, therefore decreasing the runtime. Early termination is based on the 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. Assuming that the Gaussian heuristic and the *geometric series assumption* [72] hold for a lattice, Chen [23] gives a limiting value of the root-Hermite factor δ_0 achievable by BKZ as a function of the block size k:

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

where v_k is the volume of the unit ball in dimension k. Experimental evidence suggests that we may apply the right-hand side of (1) as an estimate for δ_0 also when n is finite.

The 'lattice rule of thumb' is often used to given an approximation for δ_0 for a given k as $\delta_0 = k^{1/(2k)}$. To ease analysis, this expression, in turn, is often approximated by $\delta_0 = 2^{1/k}$ (see [75]).

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.

Assuming that (1) holds, we may conclude from Figure 1 that we do not need to consider the approximation $k^{1/(2k)}$ as it is always too pessimistic. The approximation $2^{1/k}$ 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 [41]. Computing the Voronoi cell of the lattice takes about $2^{2k+o(k)}$ operations and $2^{k+o(k)}$ memory. Sieving takes about $2^{k+o(k)}$ operations and $2^{k+o(k)}$ memory in its provable variant [1] and $2^{0.2972\,k+o(k)}$ operations and memory in its heuristic variant [49]. Enumeration can be implemented in a fashion requiring $2^{\mathcal{O}(k^2)}$ operations and

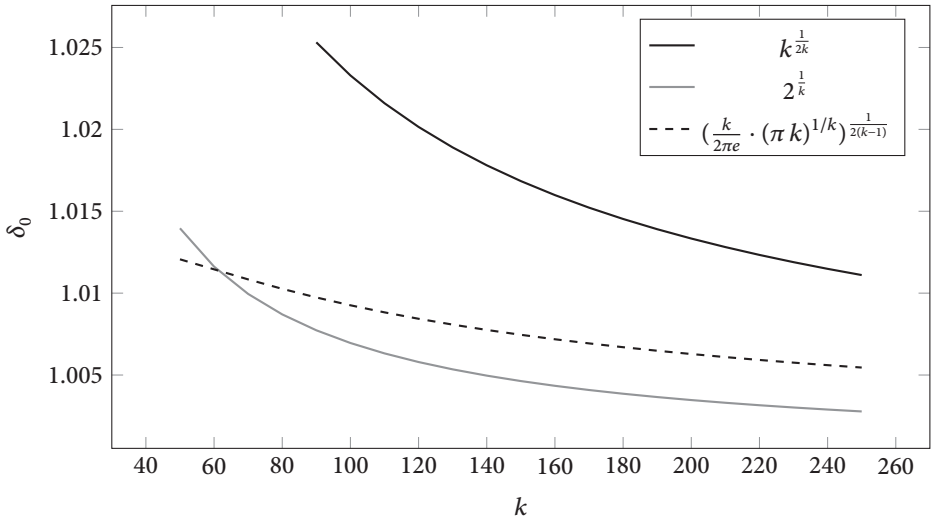

Figure 1. Estimates for δ_0 for BKZ-k.

poly(k) memory by running enumeration on an LLL-reduced lattice (Fincke–Pohst), but can be also be done in $k^{\mathcal{O}(k)}$ operations and poly(k) memory by performing heavier preprocessing on the input lattice (Kannan). Achieving $k^{\mathcal{O}(k)}$ was considered prohibitively expensive in practice until recently, but [59] proposed a variant which achieves $k^{\mathcal{O}(k)}$ with smaller overhead. Moreover, [79] showed that preprocessing local blocks with BKZ-$\mathcal{O}(k)$ before enumeration also reduces the complexity of BKZ-k to $k^{\mathcal{O}(k)}$.

Estimating ρ. No closed formula for the expected number of BKZ rounds is known. The best upper bound is exponential, but after $\rho = \frac{n^2}{k^2} \log n$ many rounds, the quality of the basis is already very close to the final output [42].

Asymptotic behaviour. Before we discuss existing estimates in the literature for the running time of BKZ, 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 3.1. *For $i \geq 1$, let $g_i(x) = x\log(g_{i-1}(x))$ with $g_0(x) = 2$. Define $g_\infty(x) = \lim_{n\to\infty} g_n(x)$. 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 $a \geq g_2(b) = b\log b$:

$$a = b\log a \quad \Rightarrow \quad a \geq b \quad \Rightarrow \quad \log a \geq \log b \quad \Rightarrow \quad a \geq b\log b$$

For the inductive step, suppose $a \geq g_i(b)$. Then $\log a \geq \log(g_i(b))$ and

$$a = b\log a \quad \Rightarrow \quad a \geq b\log(g_i(b)) = g_{i+1}(b).$$

So by induction, we have $a \geq g_n(b)$. For the second claim, if $\log a > 2$, then $b = a/\log a > 2$. 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, suppose $g_i(b) \geq g_{i-1}(b)$. Then

$$\frac{g_i(b)}{g_{i-1}(b)} \geq 1 \quad \Rightarrow \quad \log\Big(\frac{g_i(b)}{g_{i-1}(b)}\Big) \geq 0.$$

Now

$$g_{i+1}(b) - g_i(b) = b\log\Big(\frac{g_i(b)}{g_{i-1}(b)}\Big) \geq 0 \quad \Rightarrow \quad g_{i+1}(b) \geq g_i(b).$$

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 $x/\log x$ 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 $x/\log x$. Consider the solutions of the equation $x/\log x = 2$. These are precisely $x = 2$ and $x = 4$. By differentiating $x/\log x$ and evaluating at these values, and with the observation that $x/\log x$ is continuous for $x > 1$, we can see that $x/\log x$ 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, $x/\log x$ is injective here and we may conclude $g_\infty(b) = a$ as required. □

By Lemma 3.1, we have $k \geq g_n(1/(2\log\delta_0))$. In particular,

$$k \geq g_2\left(\frac{1}{2\log\delta_0}\right) = \frac{-\log(2\log\delta_0)}{2\log\delta_0}.$$

Ignoring constants, this expression simplifies to $-\log(\log\delta_0)/\log\delta_0$. Since $g_n(1/(2\log\delta_0))$ is an increasing sequence, we can lower bound the log of the time complexity of the BKZ algorithm as follows.

Corollary 3.2. *The log of the time complexity for running BKZ to achieve a root-Hermite factor δ_0 is*

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

Remark 3.3. We typically have $\log\delta_0 = O(\log n/n)$ and so since $k/\log k = 1/(2\log\delta_0)$ we have that

$$\frac{k}{\log k} = \mathcal{O}\left(\frac{n}{\log n}\right).$$

Since $x/\log x$ is injective for sufficiently large x (e.g., $x > 3$), we can conclude $k = O(n)$ for sufficiently large k and n. Therefore, in most cases considered in this work, the expressions in Corollary 3.2 could be given as $\mathcal{O}(n^2)$, $\mathcal{O}(n\log n)$ and $\mathcal{O}(n)$ because we have large n.

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

Lindner and Peikert [52] give an estimate for the runtime (in seconds) of BKZ as

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

based on experiments with the implementation of BKZ in the NTL library [74]. 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 2.3 GHz. We can hence convert this to clock cycles, giving a runtime of

$$2^{\frac{1.8}{\log\delta_0}-110}\cdot 2^{\log(2.3\cdot 10^9)} = 2^{\frac{1.8}{\log\delta_0}-110+\log(2.3\cdot 10^9)} \approx 2^{\frac{1.8}{\log\delta_0}-78.9}$$

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. Moreover, as we will show below in Section 5.3, applying this model to predict the behaviour of BKZ leads to a subexponential algorithm for solving LWE.

Albrecht et al. [6] use data points of Liu and Nguyen [54] to extrapolate a model similar to Lindner and Peikert's [52] and conclude the running time of BKZ 2.0 (in seconds) to be

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

They argue that for current implementations and estimates based on them the runtime of BKZ being non-linear in $\log \delta_0$ is more fitting than a linear model such as that of Lindner and Peikert. The analysis also gives a runtime in seconds on a 2.3 GHz 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 compared to Corollary 3.2 and assumes that enumeration in BKZ 2.0 has a complexity of $2^{\mathcal{O}(k^2)}$.

Chen and Nguyen provide a simulation algorithm for BKZ 2.0 [23, 24] 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. The simulation algorithm allows one to calculate what blocksize k will be required to obtain the approximate δ_0 given by BKZ 2.0 (cf. [24, Table 2]). Chen and Nguyen assume the SVP is solved using a pruned enumeration and they estimate the upper bound of the cost of this, for various values of k, in terms of number of nodes of the enumeration tree [24, Table 3]. The cost of BKZ is dominated by the cost of enumeration, and each round of BKZ costs "essentially $m - 1$ enumeration calls" [24] (where m is the dimension of the lattice). So the total cost of BKZ is estimated to be the number of rounds multiplied by $m - 1$ multiplied by the cost of an enumeration call.

Van de Pol and Smart [77] consider the problem from the perspective of using BKZ to solve an LWE instance or some other computational problem in lattices. They assume one has a desired level of security 2^λ (a maximum number of operations an adversary can perform) and a given lattice dimension m. These are used 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 [24, 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. The approach of van de Pol and Smart was later refined in [51].

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 [21] for dimensions up to 53 and by fitting $a\,k^2 + b\,k + c$ to the logs of these averaged running times.

The second row – labelled 'enum' – was derived by fitting $a \cdot k \log k + b\,k + c$ to [25, Table 4] (we note that these estimates were not independently verified) and assuming one enumeration costs 200 clock cycles as in [25]. We note that while BKZ 2.0 does perform local block preprocessing with BKZ-k' before calling the SVP oracle it is not clear if it as implemented in [25] achieves a complexity of $k^{\mathcal{O}(k)}$. This is because [25] does not give sufficient details to decide if preprocessing was performed with $k' = \mathcal{O}(k)$. However, from a cryptanalytical perspective it is safer to assume it achieves this bound, which is why we chose to fit the curve as we did.

The third row is based on [48] which reports a time complexity of $2^{0.3366\,k + o(k)}$ asymptotically and $2^{0.45\,k - 19}$ seconds in practical experiments on a 2.66 GHz CPU for small k. Note that the leading coefficient in these experiments is bigger than 0.3366 from the asymptotic statement because of the $+o(k)$ term in the asymptotic

name	data source	$\log t_k$
fplll	fplll 4.0.4	$0.0135\,k^2 - 0.2825\,k + 21.02$
enum	[25]	$0.270189\,k \log k - 1.0192\,k + 16.10$
sieve	[48]	$0.3366\,k + 12.31$

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

expression. We brush over this difference and simply estimate the cost of sieving as $2^{0.3366\,k+c}$ operations where we derive the additive constant c from timings derived from practical experiments in [48]. We note that [49] does not include any experimental results. Hence, we cannot estimate when the cost hidden in the $+o(k)$ term is small enough. Indeed, estimating the cost of sieving in practice is stressed as an open problem in [49].

To fit curves we used Sage's function `find_fit` [76] which calls SciPy's function `scipy.optimize.leastsq` [44] which in turn uses MINPACK's `lmdif` [60].

Overall. By setting $\rho = \frac{n^2}{k^2}\log n$, 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 [74] and fplll [21]. Neither of these implementations incorporate all techniques which are collectively known as BKZ 2.0. Hence, both implementations have a complexity of $2^{\mathcal{O}(k^2)}$. However, the next version of fplll implements recursive local block preprocessing with BKZ [5].

3.3 Choosing the number of samples 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 [58]. This 'optimal subdimension' is also often heuristically chosen even where the above relation between volume and dimension does not hold. In [77] van de Pol and Smart 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)

In order 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 $L = \{\mathbf{w} \in \mathbb{Z}_q^m \mid \mathbf{wA} \equiv 0 \bmod q\}$, which is exactly solving the *short integer solutions* (SIS) problem [2]. Consider $\langle \mathbf{v}, \mathbf{c} \rangle$. If $\mathbf{c} = \mathbf{As} + \mathbf{e}$ 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 4.1 ([52]). *Given an LWE instance characterised by n, α, q and a vector $\mathbf{v}$ of length $\|\mathbf{v}\|$ in the scaled dual lattice $L = \{\mathbf{w} \in \mathbb{Z}_q^m \mid \mathbf{wA} \equiv 0 \bmod q\}$, the advantage of distinguishing $\langle \mathbf{v}, \mathbf{e} \rangle$ from random is close to $\exp(-\pi(\|\mathbf{v}\| \cdot \alpha)^2)$.*

Remark 4.2. For example, Stehlé [75] 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 4.1, this results in an advantage 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 distinguishing advantage ε, but then running the algorithm about $1/\varepsilon^2$ times will achieve a success probability close to 1 by the Chernoff bound [26]. 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 4.3. *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 $\mathbf{v}$ of norm $\|\mathbf{v}\| = \frac{1}{\alpha}\sqrt{\ln(1/\varepsilon)/\pi}$.*

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 $f(\varepsilon)$ denote $\sqrt{\ln(1/\varepsilon)/\pi}$.

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, $3\alpha q/\sqrt{2\pi}$) 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.)

Again, depending on the algorithm, it may be advantageous to accept a lower success probability ε. Then, approximately $\log(1-\varepsilon')/\log(1-\varepsilon)$ iterations will achieve a success probability close to the target ε', since $\varepsilon' = 1 - (1-\varepsilon)^m$.

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. For example, the Arora–Ge algorithm (cf. Section 5.6) directly recovers $\mathbf{s}$.

5 Algorithms

5.1 Exhaustive search

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

Theorem 5.1. *The time complexity of solving Search-LWE with success probability ε with exhaustive search is $m \cdot (2t\alpha q + 1)^n \cdot 2n = 2^{n \log (2t\alpha q+1)+\log n+1+\log m}$. The memory complexity is n, the sample complexity is $n + m$ with*

$$m = \frac{\log(1-\varepsilon) - n\log(2t\alpha q + 1)}{\log(2t\alpha)}$$

for some small parameter $t = \omega(\sqrt{\log n})$.

Proof. Consider $\{-t\alpha q, \ldots, t\alpha q\}$ for $t = \omega(\sqrt{\log n})$. By Lemma 2.4, an LWE sample has error which falls in this range with overwhelming probability. Apply Lemma 2.1 to obtain an LWE instance with $\mathbf{s}_{(i)} \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\alpha q}$, i.e. the secret is distributed the same as the error. We are therefore able to estimate the size of each component of the secret as $|\mathbf{s}_{(i)}| \leq t\alpha q$. Therefore, to check all possible secrets we must enumerate approximately $(2t\alpha q + 1)^n$ vectors. For each vector we perform about $2n$ operations in $\mathbb{Z}_q$ when computing the inner product.

We need n samples to apply Lemma 2.1 (if these n samples do not have full rank, pick n samples again from the overall set of all samples). We know that the correct $\mathbf{s}$ will produce $e_i = \langle \mathbf{a}_i, \mathbf{s}\rangle - c_i$ with $e_i \in \{-t\alpha q, \ldots, t\alpha q\}$ with overwhelming probability. Wrong guesses $\mathbf{s}'$ will produce random elements in $\mathbb{Z}_q$ which land within the acceptable range with probability $\leq (\lceil 2t\alpha q\rceil + 1)/q \approx 2t\alpha$. For the wrong guess $\mathbf{s}'$ to pass the test it must pass for all m samples, which happens with probability $(2t\alpha)^m$. There are $(2t\alpha q + 1)^n - 1$ wrong choices for $\mathbf{s}$. By the union bound, we will hence accept a false positive with probability $p_f \leq (2t\alpha)^m \cdot (2t\alpha q + 1)^n$. Choosing

$$m \geq \frac{\log p_f - n\log(2t\alpha q + 1)}{\log(2t\alpha)}$$

this happens with a probability $\leq p_f$. Picking $p_f = 1 - \varepsilon$ to ensure that p_f is sufficiently small finishes the proof. □

Corollary 5.2. *Let $q = n^c$ and $\alpha q = \sqrt{n}$. Then the time complexity of solving Search-LWE with success probability ε with exhaustive search is*

$$2^{n \log (2t\sqrt{n}+1)+\log n+1+\log m}.$$

The memory complexity is n. The sample complexity is $m + n$ with

$$m = \frac{\log(1-\varepsilon) - n\log(2t\sqrt{n} + 1)}{(\frac{1}{2} - c)\log n + \log(2t)}.$$

Remark 5.3. The complexity is independent of α and q but depends on their product αq and n.

Meet-in-the-middle. As mentioned in [14] there is also a *meet-in-the-middle* (MITM) algorithm. MITM 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 5.4. *Let an LWE instance be parametrised by n, α, q. If there are $n + m$ samples satisfying $2t\alpha m < 1/C$ for some constant $C > 1$ and $(2t\alpha)^m \cdot (2t\alpha q + 1)^{n/2} = \mathrm{poly}(n)$ for some small parameter $t = \omega(\sqrt{\log n})$, then there is an MITM algorithm which solves Search-LWE with non-negligible probability which runs in time*

$$\mathcal{O}\big(m(2t\alpha q + 1)^{n/2}(2n + (n/2 + \mathrm{poly}(n)) \cdot \log(m(2t\alpha q + 1)))\big)$$

and requires memory $m \cdot (2t\alpha q + 1)^{n/2}$.

Proof. Consider $\{-t\alpha q, \ldots, t\alpha q\}$ for $t = \omega(\sqrt{\log n})$. By Lemma 2.4, an LWE sample has error which falls in this range with overwhelming probability. Apply Lemma 2.1 to obtain an LWE instance with $\mathbf{s}_{(j)} \leftarrow_\$ \mathcal{D}_{\mathbb{Z},\alpha q}$, i.e.

the secret is distributed the same as the error. This costs n samples. Given m samples $(\mathbf{a}_k, \langle \mathbf{a}_k, \mathbf{s}\rangle + e_k)$, split $\mathbf{a}_k = \mathbf{a}_k^l \parallel \mathbf{a}_k^r$ in half and for each possibility $\mathbf{s}_i^l$ of the first half of $\mathbf{s}$ compute the inner product of the first half of $\mathbf{a}_k$ and $\mathbf{s}_i^l$. Let the output of guess $\mathbf{s}_i^l \in \mathbb{Z}_q^{n/2}$ for each of the m samples be

$$\mathbf{u}_{\mathbf{s}_i^l} = \left(\langle \mathbf{a}_0^l, \mathbf{s}_i^l\rangle, \ldots, \langle \mathbf{a}_{m-1}^l, \mathbf{s}_i^l\rangle\right).$$

Store a table T whose entries map vectors $\mathbf{u}_{\mathbf{s}_i^l}$ to $\mathbf{s}_i^l$. Generating the table costs $m \cdot 2n \cdot (2t\alpha q + 1)^{n/2}$ operations, since we estimate the size of each component of the secret as $|\mathbf{s}_{(i)}| \leq t\alpha q$ and hence we expect $(2t\alpha q + 1)^{n/2}$ candidates $\mathbf{s}_i^l$ and for each of these we calculate m inner products. Sort the table into lexicographical ordering component-wise. This costs

$$\mathcal{O}\left(m(2t\alpha q + 1)^{n/2} \cdot n/2 \cdot \log\left(m(2t\alpha q + 1)\right)\right)$$

operations.

Now, for each candidate $\mathbf{s}_j^r \in \mathbb{Z}_q^{n/2}$ for the second half of the secret compute the inner product of the second half of each $\mathbf{a}_k$ with $\mathbf{s}_j^r$ and then subtract from c_k, to obtain the vector

$$\mathbf{v}_{\mathbf{s}_j^r} = \left(c_0 - \langle \mathbf{a}_0^r, \mathbf{s}_j^r\rangle, \ldots, c_{m-1} - \langle \mathbf{a}_{m-1}^r, \mathbf{s}_j^r\rangle\right).$$

Query T with $\mathbf{v}_{\mathbf{s}_j^r}$; that is, view T as a list and sort $\mathbf{v}_{\mathbf{s}_j^r}$ into this list. We can do this in

$$\log |T| = n/2 \cdot \log\left(m(2t\alpha q + 1)\right)$$

operations per candidate by binary search. Since there are $(2t\alpha q + 1)^{n/2}$ possible second halves of the secret $\mathbf{s}_j^r$, the overall cost of querying is $(2t\alpha q + 1)^{n/2} \cdot n/2 \cdot \log\left(m(2t\alpha q + 1)\right)$. When we have sorted $\mathbf{v}_{\mathbf{s}_j^r}$ into the list, we return which vectors $\mathbf{u}_{\mathbf{s}_i^l}$ it has fallen between, and check if they are near to $\mathbf{v}_{\mathbf{s}_j^r}$ in Euclidean distance (we specify near in a moment).

If the vector $\mathbf{v}_{\mathbf{s}_j^r}$ is near to the vector $\mathbf{u}_{\mathbf{s}_i^l}$ return $\mathbf{s}_i^l$ and treat $\mathbf{s}_i^l \parallel \mathbf{s}_j^r$ as a candidate secret, and check if it is correct. If this process returns no secret overall, get fresh samples and start over.

If $\mathbf{s} = \mathbf{s}_{i'}^l \parallel \mathbf{s}_{i'}^r$ is the correct secret then

$$\mathbf{v}_{\mathbf{s}_{i'}^r} - \mathbf{u}_{\mathbf{s}_{i'}^l} = (e_0, \ldots, e_{m-1}) \bmod q.$$

With overwhelming probability we will have $e_k \in \{-t\alpha q, \ldots, t\alpha q\}$ for $0 \leq k < m$. Therefore

$$\|\mathbf{v}_{\mathbf{s}_{i'}^r} - \mathbf{u}_{\mathbf{s}_{i'}^l}\| = \|(e_0, \ldots, e_{m-1})\| = \sqrt{m}(t\alpha q).$$

A candidate should be rejected if the distance between $\mathbf{u}_{\mathbf{s}_i^l}$ and $\mathbf{v}_{\mathbf{s}_j^r}$ is more than $\sqrt{m}(t\alpha q)$ and should be accepted otherwise.

This means that with overwhelming probability the algorithm will identify the correct secret as long as the error does not cause a wrap around mod q on any component. That is, over the integers we have

$$\mathbf{v}_{\mathbf{s}_j^r} - \mathbf{u}_{\mathbf{s}_i^l} = (e_0, \ldots, e_{m-1}) + (qh_0, \ldots, qh_{m-1})$$

for constants $h_k \in \{-1, 0, 1\}$. A wrap around mod q on one component corresponds to $h_k = \pm 1$ on that component, but we require that $h_k = 0$ on all components (for our lexicographical ordering to work). The above is equivalent on each component to $c_k = b_k + e_k + qh_k$ where $b_k = \langle \mathbf{a}_k, \mathbf{s}\rangle \bmod q$. We have that $h_k = \pm 1$ will not occur whenever b_k is not in a band of width $t\alpha q$ either side of q, but may occur otherwise. So we can bound the probability that a correct secret is rejected because of a wrap around error by the probability that at least one of the m components has $b_k \in [0, t\alpha q] \cup [q - t\alpha q, q)$. The probability that one component has $b_k \in [0, t\alpha q] \cup [q - t\alpha q, q)$ is $2t\alpha q/q = 2t\alpha$ so by the union bound the probability that at least one of the m components has $b_k \in [0, t\alpha q] \cup [q - t\alpha q, q)$ is $\leq m \cdot 2t\alpha$. We want to bound m so that this event only happens with probability at most $1/C$ for some constant C, i.e. $(2t\alpha m) < 1/C$.

Consider now the chance of a false positive, i.e. a wrong candidate secret $\mathbf{s}_i^l$ being suggested for some candidate $\mathbf{s}_j^r$. Since $\mathbf{a}_k$ is uniformly random, for any $\mathbf{s}_i^l$, we have that $\mathbf{u}_{\mathbf{s}_i^l}$ is essentially a random vector where each component takes one of q values. The chance of a wrong candidate $\mathbf{s}_j^r$ producing a $\mathbf{v}_{\mathbf{s}_j^r}$ matching to a

given $\mathbf{u}_{\mathbf{s}_i^l}$ to within distance $\sqrt{m}(t\alpha q)$ is the chance of getting to within $\pm t\alpha q$ on every component. Therefore the chance of a false positive is $((\lceil 2t\alpha q\rceil + 1)/q)^m \approx (2t\alpha)^m$ There are $(2t\alpha q + 1)^{n/2} - 1$ wrong choices for $\mathbf{s}_i^l$. We hence expect to test $(2t\alpha)^m \cdot (2t\alpha q + 1)^{n/2}$ candidates per $\mathbf{s}_j^r$ and thus require

$$(2t\alpha)^m \cdot (2t\alpha q + 1)^{n/2} = \mathrm{poly}(n). \qquad \square$$

Remark 5.5. If $q = n^c$ and $\alpha q = \sqrt{n}$, then setting $m = n$ satisfies Theorem 5.4 for $c \geq 2$ and $t = 2\sqrt{\log n}$.

5.2 BKW

The BKW (Blum, Kalai, Wasserman) algorithm was introduced in [16] and shows that subexponential algorithms exist for learning parity functions in the presence of noise: the BKW algorithm solves the LPN problem (learning parity with noise) in time $2^{\mathcal{O}(n/\log n)}$. BKW can be adapted to solve LWE [70] and the complexity of this has been studied in [6]. 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.[1]

Theorem 5.6 ([6]). *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, $0 < \varepsilon < 1$ 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*

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

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

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

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

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

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

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

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

1 Albrecht et al. [6] optimistically give $m = \varepsilon/\exp(-\pi\,\alpha^2\, 2^a)$, but by the Chernoff bound we need about $1/\exp(-\pi\,\alpha^2\, 2^a)^2$ samples to distinguish.

To pick a and b, recall from Remark 4.2 that in order to distinguish $L_{\mathbf{s},\chi}$ from random using SIS an appropriately short choice for $\mathbf{v}_i$ is $\|\mathbf{v}_i\| \cdot \alpha q = \sqrt{2^a} \cdot \alpha q \leq q$ hence a suitable choice for a is $a \leq \log(\alpha^{-2})$.

Corollary 5.7 ([6]). *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*

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

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

Specialising Corollary 5.7 with $q = n^c$ and $\alpha q = \sqrt{n}$ we get:

Corollary 5.8. *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*

$$\left(\frac{q^b - 1}{2}\right) \cdot \left(\frac{a(a-1)}{2} \cdot (n+1)\right) + \mathrm{poly}(n) \leq q^b \cdot (a^2 n) + \mathrm{poly}(n) = 2^{\frac{n}{2-(1/c)}} \cdot \mathrm{poly}(n)$$

operations in $\mathbb{Z}_q$.

Remark 5.9. 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.

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 $m = \mathrm{poly}(n)$, 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

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

By balancing both sides we may reduce the complexity of the BKW algorithm to

$$\mathcal{O}(2^{(cn\log n/(2c\log n - d))}) = 2^{n/(2-o(1))}$$

for some constant d. This can make a significant difference for picking concrete parameters.

Example 5.10. 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 5.7. Balancing the two steps of the algorithm would reduce this to 2^{171} operations in $\mathbb{Z}_q$.

The search variant of BKW algorithm as given in [6] was later improved by Duc, Tramèr and Vaudenay [27] by using a discrete Fourier transform to recover a correct subset of components of $\mathbf{s}$. In particular, Duc et al. arrive at the following complexity result.

Theorem 5.11 ([27]). *Let an LWE instance be parametrised by n, α, q and let $a, b \in \mathbb{N}$ be such that $a \cdot b = n$. Let C_{FFT} be the small constant in the complexity of the fast Fourier transform computation. Let $0 < \varepsilon < 1$ be a targeted success rate and define $\varepsilon' = (1-\varepsilon)/a$. For $0 \leq j \leq a-1$ let*

$$m_{j,\varepsilon} = 8 \cdot b \cdot \log\frac{q}{\varepsilon}\left(1 - \frac{2\pi^2\sigma^2}{q^2}\right)^{-2^{a-j}}.$$

The time complexity to recover the secret $\mathbf{s}$ with probability at least ε is $c_1 + c_2 + c_3 + c_4$ where

$$c_1 = \frac{q^b - 1}{2} \cdot \left(\frac{(a-1)(a-2)}{2}(k+1) - \frac{b}{6}\big(a(a-1)(a-2)\big)\right)$$

is the number of additions in $\mathbb{Z}_q$ to produce tables,

$$c_2 = \sum_{j=0}^{a-1} m_{j,\varepsilon'} \cdot \frac{a-1-j}{2}(k+2)$$

is the number of additions in $\mathbb{Z}_q$ to recover $\mathbf{s}$*,*

$$c_3 = 2\Big(\sum_{j=0}^{a-1} m_{j,\varepsilon'}\Big) + C_{\mathrm{FFT}} \cdot k \cdot q^b \cdot \log q$$

is the number of operations in $\mathbb{C}$ to prepare and compute the discrete Fourier transforms, and

$$c_4 = (a-1)(a-2) \cdot b \cdot \frac{q^b - 1}{2}$$

is the number of operations in $\mathbb{Z}_q$ for back substitution. Furthermore we require $(a-1)\frac{q^b-1}{2} + m_{0,\varepsilon}$ calls to $L_{\mathbf{s},\chi}$ and storage for

$$\Big(\frac{q^b-1}{2}(a-1)\Big(k+1-b\frac{a-2}{2}\Big) + m_{0,\varepsilon}\Big)$$

elements in $\mathbb{Z}_q$ and q^b elements in $\mathbb{C}$.

A reference implementation of the BKW algorithm for LWE as described in [6] is available as [3].

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 $L = \{\mathbf{w} \in \mathbb{Z}_q^m \mid \mathbf{wA} \equiv 0 \bmod q\}$. 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 $\mathrm{vol}(L) = q^n$ (see [58]).

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 $\mathbf{b}_0\,\mathbf{A} = 0 \bmod q$. 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 5.12. *Let an LWE instance be parametrised by n, α, q. Any lattice reduction algorithm achieving log root-Hermite factor*

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

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 $\delta_0^m = \|\mathbf{v}\|/\mathrm{vol}(L)^{\frac{1}{m}}$ so we have $\|\mathbf{v}\| = \delta_0^m\, q^{\frac{n}{m}}$. On the other hand, we require $\|\mathbf{v}\| = \frac{1}{\alpha} f(\varepsilon)$ by Corollary 4.3. By Section 3.3 the optimal subdimension m which minimises the quantity $\delta_0^m q^{\frac{n}{m}}$ is $m = \sqrt{n \log q / \log \delta_0}$. 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

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

as our desired log root-Hermite factor. □

Corollary 5.13. *Given an LWE instance parametrised by n, $q = n^c$, $\alpha q = \sqrt{n}$. Any lattice reduction algorithm achieving log root-Hermite factor*

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

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

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

Table 2. Time complexity for distinguishing $L_{\mathbf{s},\chi}$ from random with advantage $\varepsilon \approx 1/23$ based on lattice reduction estimates from the literature. The last four rows give estimates for $q = n^c$ and $\alpha = n^{1/2-c}$.

Proof. We have

$$\begin{aligned}
\delta_0^m q^{\frac{n}{m}} &= \|\mathbf{v}\| \\
\delta_0^m n^{\frac{cn}{m}} &= n^{c-\frac{1}{2}} f(\varepsilon) \\
\sqrt{\frac{cn\log n}{\log\delta_0}}\log\delta_0 + \frac{cn}{\sqrt{cn\log n/\log\delta_0}}\log n &= \left(c-\frac{1}{2}\right)\log n + \log f(\varepsilon) \\
\frac{cn\log n\log\delta}{\log\delta_0} + cn\log n &= \sqrt{\frac{cn\log n}{\log\delta_0}}\left(\left(c-\frac{1}{2}\right)\log n + \log f(\varepsilon)\right) \\
2cn\log n &= \sqrt{\frac{cn\log n}{\log\delta_0}}\left(\left(c-\frac{1}{2}\right)\log n + \log f(\varepsilon)\right) \\
\frac{2cn\log n}{((c-\frac{1}{2})\log n + \log f(\varepsilon))} &= \sqrt{\frac{cn\log n}{\log\delta_0}} \\
\frac{(2cn\log n)^2}{((c-\frac{1}{2})\log n + \log f(\varepsilon))^2} &= \frac{cn\log n}{\log\delta_0} \\
\frac{4cn\log n}{((c-\frac{1}{2})\log n + \log f(\varepsilon))^2} &= \frac{1}{\log\delta_0} \\
\frac{((c-\frac{1}{2})\log n + \log f(\varepsilon))^2}{4cn\log n} &= \log\delta_0.
\end{aligned}$$

□

Remark 5.14. 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 it 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 [35] (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 $f(\varepsilon) = 1$, i.e. $\varepsilon \approx 1/23$.

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 considered not to 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 about $1/(\varepsilon')^2$ times to boost the overall success probability to a success probability close to 1.

Example 5.15. Setting $n = 256$, $q = 65537$ and $\sigma = \alpha q/\sqrt{2\pi} \approx 25.53$ following [70] and picking $\varepsilon = 0.1$ we get a target $\delta_0 = 1.0040513$ by Lemma 5.12 and thus $m = 838$. Computing the expected number of clock cycles, we end up with the estimates shown in Table 3. In contrast, picking ε' such that $1/(\varepsilon')^2$ multiplied by the solving time is minimised, we get the estimates shown in Table 4.

model	block size k	log clock cycles
enum	391	270
sieve	391	182

Table 3. Estimated clock cycles needed to mount a lattice reduction attack, parametrised as above, with $\varepsilon = 0.1$.

model	$\log \varepsilon'$	δ_0	block size k	log clock cycles
enum	−21	1.005232	267	204
sieve	−11	1.004801	304	164

Table 4. Estimated clock cycles for an optimised lattice reduction attack, with ε' chosen to minimise the solving time.

5.4 Decoding approach

This approach solves LWE by solving the BDD problem (cf. [52]). The most basic way of solving a BDD instance is using Babai's nearest plane algorithm [13]. This approach 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 [52].

The probability the nearest plane 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 plane. Lindner and Peikert [52] 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

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

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 $\operatorname{erf}(2x)/\operatorname{erf}(x) > \operatorname{erf}(2y)/\operatorname{erf}(y)$ for all $0 < x < y$, 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 $t_{\mathrm{NP}}(\delta_0, \varepsilon) = t_{\mathrm{node}} \cdot \prod_{i=0}^{m-1} d_i$ such that equation (2) 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 approach to achieve a success probability ε could be determined as

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

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

We note that in [52, 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 approach is less than an equivalent distinguishing approach (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 we may opt to run the algorithm many times with a lower advantage. This typically reduces the overall complexity.

Solving BDD by enumeration: An update (Liu, Nguyen). Liu and Nguyen [54] 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 [30]. 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 $\zeta_i(\mathbf{x}) = \langle \mathbf{x}, \mathbf{b}_i^* \rangle / \|\mathbf{b}_i^*\|$. 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 $\|\mathbf{b}_i^*\|$.

Due to these similarities between the Lindner–Peikert method and pruning techniques, Liu and Nguyen implement a variant of the Lindner–Peikert 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 [30], and uses linear pruning in which the bounds $R_k = \sqrt{k/m}R_m$ are used. Over the same parameters used in [52], this linear pruning is shown to improve on both the original nearest plane algorithm and the improved variant.

Runtime analysis. In any lattice decoding approach, 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 approach, 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

$$t_{\mathrm{node}} = 2^{-16} \cdot 2.33 \cdot 10^9 \approx 2^{15.1}$$

clock cycles, whereas [30] 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 $t_{\mathrm{node}} = 2^{15.1}$.

Calculating the success probability is harder. For nearest plane, we can use equation (2), 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 and Peikert's strategy of increasing d_i one by one. For enumeration and pruning, we need to use the method as used in [54], 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 approach 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 5.16. For $n = 192$, $q = 4093$, $\alpha q = 8.87$, Lindner and Peikert [52] 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 approach using a lattice reduction with $\delta_0 = 1.0077$ and $\varepsilon = 2^{-12}$. This lattice reduction takes $2^{65.6}$ seconds in the Lindner–Peikert model for lattice reduction. Our estimator suggests $\varepsilon = 2^{-15}$ and $\delta_0 = 1.0077566$ which implies a lattice reduction cost of $2^{66.48}$ 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 [54] but we restrict our attention to [52] which gives comparable results and is easier to estimate.

5.5 Reducing BDD to uSVP

Albrecht, Fitzpatrick and Göpfert [10] consider the complexity of solving LWE by reducing BDD to uSVP (unique shortest vector problem). Formally, the γ-uSVP problem is as follows: given a lattice L such that $\lambda_2(L) > \gamma\lambda_1(L)$, find a shortest non-zero vector in L.

To reduce BDD to uSVP Kannan's embedding technique [46] is used. The idea is to embed the lattice $L(\mathbf{A}) = \{\mathbf{A}\mathbf{u} \mid \mathbf{u} \in \mathbb{Z}_q^n\}$ generated by the columns of the LWE instance (and our usual lattice for consideration when 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, $\mathrm{dist}(\mathbf{x}, L)$, as this length. If the embedding factor $t = \mathrm{dist}(\mathbf{c}, L(\mathbf{A})) < \lambda_1(L(\mathbf{A}))/(2\gamma)$ then $L(\mathbf{B})$ contains a γ-unique shortest vector, $\mathbf{c}' = (\mathbf{e}, -t)$ (see [56]), from which we can take the first m components to recover $\mathbf{e}$, hence solving the BDD instance.

To solve a γ-uSVP instance, we may reduce the problem to κ-HSVP (Hermite shortest vector problem). Let $\gamma = \kappa^2$. Lovász [55] 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. Ling et al. [53] show that whenever $\kappa > \sqrt{N}$, for N the dimension of the lattice, this result can be improved. They show that 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 (see [29]). The value τ is taken to be a constant, which is experimentally derived in [29] 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 m and τ, which depend 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 [10].[2]

Lemma 5.17 ([10, Lemma 2]). *Let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, let $\alpha q > 0$ and let $\varepsilon' > 1$. Let $\mathbf{e} \in \mathbb{Z}_q^m$ such that each component is drawn from χ and considered mod q. Assuming the Gaussian heuristic for $L(\mathbf{A})$, i.e.*

$$\lambda_1(L(\mathbf{A})) \geq \sqrt{m/(2\pi e)}\,\mathrm{vol}(L)^{1/m},$$

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

$$\frac{\min\{q, q^{1-\frac{n}{m}}\sqrt{m/(2\pi e)}\}}{\varepsilon' s\sqrt{m}/\sqrt{\pi}}$$

with probability greater than $1 - (\varepsilon' \cdot \exp(\frac{1-\varepsilon'^2}{2}))^m$.

Hence, setting $t = \|\mathbf{e}\|$, $\mathbf{B}$ is a basis of a lattice whose gap is determined by Lemma 5.17. Using Lemma 5.17 and under the assumption $q^{1-\frac{n}{m}}\sqrt{m/(2\pi e)} < q$, we require a gap of size approximately

$$\frac{\lambda_2}{\lambda_1} = \frac{q^{1-\frac{n}{m}}\sqrt{1/(2e)}}{\varepsilon'\alpha q}$$

and so we require a δ_0 determined by $q^{1-\frac{n}{m}}\sqrt{1/(2e)} \geq \tau\delta_0^m\,\varepsilon'\alpha q$. The determination of τ is discussed at length in [10] but we have that $\tau \leq 0.4$ (depending on the algorithm) for a success probability of 0.1 based on the experimental results in [10, 29]. We stress that no data is publicly available on τ for smaller success probabilities. In [10] it is shown that for a fixed δ_0 the optimal subdimension is $m = \sqrt{n\log q/\log\delta_0}$ as in Section 3.3. We may use this to determine δ_0 using the expression above (where for simplicity we assume equality).

Lemma 5.18. *Given an LWE instance characterised by n, α, q. Any lattice reduction algorithm achieving log root-Hermite factor*

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

solves LWE with success probability greater than

$$\varepsilon_\tau \cdot \left(1 - \left(\varepsilon' \cdot \exp\left(\frac{1-\varepsilon'^2}{2}\right)\right)^m\right)$$

for some $\varepsilon' > 1$ and some fixed $\tau \leq 1$, and $0 < \varepsilon_\tau < 1$ 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:

$$q^{1-\frac{n}{\sqrt{n\log q/\log\delta_0}}}\sqrt{\frac{1}{2e}} = \varepsilon'\tau\,\alpha q\,\delta_0^{\sqrt{n\log q/\log\delta_0}}$$

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

Rearranging, we obtain

$$\left(1 - \frac{n}{\sqrt{n \log q / \log \delta_0}}\right) \log q + \log \sqrt{\frac{1}{2e}} = \log(\varepsilon' \tau \alpha q) + \sqrt{\frac{n \log q}{\log \delta_0}} \log \delta_0$$

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

$$\left(\frac{\sqrt{n \log q / \log \delta_0} - n}{\sqrt{n \log q / \log \delta_0}}\right) \log q - \sqrt{\frac{n \log q}{\log \delta_0}} \log \delta_0 = \log(\varepsilon' \tau \alpha q \sqrt{2e})$$

$$\sqrt{\frac{n \log q}{\log \delta_0}} \log(\varepsilon' \tau \alpha \sqrt{2e}) = 2n \log q$$

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

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

Finally, the success probability is computed as the probability the gap is as required in Lemma 5.17 multiplied by the success probability of our algorithm ε_τ. □

Corollary 5.19. *Given an LWE instance characterised by n, $q = n^c$, $\alpha q = \sqrt{n}$. Any lattice reduction algorithm achieving log root-Hermite factor*

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

solves LWE with success probability greater than

$$\varepsilon_\tau \cdot \left(1 - \left(\varepsilon' \cdot \exp\left(\frac{1 - \varepsilon'^2}{2}\right)\right)^m\right)$$

for some $\varepsilon' \approx 1$ and some fixed $\tau \leq 1$ and $0 < \varepsilon_\tau < 1$ as a function of τ.

Comparing Corollary 5.19 with Corollary 5.13 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 $\log(1/(\tau\sqrt{2e})) > \log(f(\varepsilon))$ under the condition that $\varepsilon \approx \varepsilon_\tau$ so that the success probabilities are equal in both cases.

However, using Kannan's embedding is not necessarily more efficient than the decoding approach discussed in Section 5.4, as the following example highlights.

Example 5.20. Letting $n = 256$, $q = 65537$ and $\sigma = \alpha q / \sqrt{2\pi} \approx 25.53$ following [70] and choosing $t = 1$, $\tau' = 0.310$ and $\varepsilon' = 1.01$ we get a target $\delta_0 = 1.004634$ by Lemma 5.18 and thus $m = 784$. According to [10] we have $\varepsilon_\tau = 0.1$. Our estimator predicts block size $k = 321$ and 2^{205} clock cycles for performing this lattice reduction. In contrast, performing the decoding approach from Section 5.4 is predicted to cost 2^{172} clock cycles overall to perform lattice reduction which achieves root-Hermite factor $\delta_0 = 1.005198$ and to run the final decoding stage.

Finally, suppose $t < \|\mathbf{e}\|$. In this case no efficient method for determining λ_2/λ_1 is known. The assumption in [10] 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, denoted τ', is introduced which relates to the gap from the case $t = e$ when actually computing with $t = 1$. Setting $t = 1$ is typically more efficient than $t = \|\mathbf{e}\|$ and we have $\tau' \approx 0.3$, see [10] for details.

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 [12]. This approach solves for $\mathbf{s}$ directly.

In particular, [12] 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 (cf. Lemma 2.4).

Polynomials are constructed from the observation that the error, when falling in this range, is always a root of the polynomial $P(x) = x\prod_{i=1}^{t}(x+i)(x-i)$. Then, we know the secret $\mathbf{s}$ is a root of $P(\mathbf{a}\cdot\mathbf{x} - c)$ constructed from LWE samples. In the Arora–Ge algorithm the system of non-linear equations constructed this way is solved by replacing each monomial with a new variable and solving the resulting linear system. However, this means that we need $\mathcal{O}(n^{2t+1})$ 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 5.21 ([7, Theorem 5]). *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

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

and memory complexity

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

Remark 5.22. Specialising to $\sigma = \sqrt{n}$ the complexity is $\mathcal{O}(2^{(2+\varepsilon)\omega n \log\log n})$. In this case, the Arora–Ge algorithm is asymptotically slower than the BKW algorithm and lattice reduction if sieving is used to implement the SVP oracle, but asymptotically faster than lattice reduction if enumeration is used to implement the SVP oracle.

This can be improved by using Gröbner basis techniques [7]. In particular, to solve via linearisation as in [12], we require $\mathcal{O}(n^d)$ 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

$$\mathcal{O}\left(mD_{\mathrm{reg}}\binom{n + D_{\mathrm{reg}}}{D_{\mathrm{reg}}}^{\omega}\right),$$

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. A semi-regular sequence of m polynomials of degree d in n variables is a sequences with the following Hilbert series:

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

It is assumed that random systems behave like semi-regular sequences. A justification as to why this is a reasonable assumption is given in [7]. 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 5.23 ([7]). *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 $\mathcal{O}(2^{2.35\omega n + 1.13n})$, memory complexity $\mathcal{O}(2^{5.85n})$ and sample complexity $m = \exp(\frac{\pi}{4} \cdot n)$.*

Hence, for $\alpha q = \sqrt{n}$ applying Gröbner basis algorithms is in the same complexity class as the BKW algorithm or lattice reduction when sieving implements the SVP oracle, albeit with a larger leading constant in the exponent.

Remark 5.24. The complexity 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 note that there is a gap between security reductions and the best known algorithms for solving LWE with binary secrets. On the one hand, theoretical results show that for an LWE instance with a binary secret to be as hard as general LWE in dimension n a dimension of $n \log q$ is sufficient [18, 57]. On the other hand, the best known algorithms for solving LWE with a binary secret from [14] manages to solve LWE instances with a binary secret and in dimension $n \log \log n$ in about the same complexity as it would take to solve a standard LWE instance in dimension n. Hence, based on the currently best known attacks we would conclude that we only need to increase the dimension to $n \log \log n$ instead of $n \log q$. Hence, there is room for improvement either for algorithms or for security reductions.

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 $t\alpha q$, for some small parameter $t = \omega(\sqrt{\log n})$, 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 5.1, exhaustive search will take time $m \cdot 3^n \cdot (2n) = 2^{n \log 3 + \log n + 1 + \log m}$ if $\mathbf{s}_{(i)} \in \{-1, 0, 1\}$.

As observed in [14] we can also combine exhaustive search with other algorithms to improve the complexity by guessing, say, g components of the secret and then running our algorithm on the reduced small secret LWE instance of dimension $n - g$. With this strategy any algorithm discussed below can be turned into an algorithm which has at most the cost of exhaustive search.

MITM. By exactly the same argument as in Theorem 5.4, 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 $\mathcal{O}(3^{n/2})$ and require $\mathrm{poly}(n) \cdot 3^{n/2}$ memory.

6.2 Modulus switching for lattice reduction

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 at random from $\mathbb{Z}_q$. After modulus switching, the transformed instance has an 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, when we have a small secret we may obtain a speed up by modulus switching before performing lattice reduction (for example, as described in Sections 5.3, 5.4 and 5.5).

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 $\|\mathbf{v}\| = \frac{1}{\sqrt{2\alpha}} f(\varepsilon)$.

Lemma 6.1. *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*

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

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 2.2, 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 5.12 we require

$$\delta_0 = 2^{(\log(\sqrt{2}\alpha \frac{1}{f(\varepsilon)}))^2/(4n \log p)}.$$

□

Corollary 6.2. *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*

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

Proof. From Lemma 2.2 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 6.1 we have

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

□

Example 6.3. Setting $n = 256$, $q = 65537$ and $\sigma = \alpha q/\sqrt{2\pi} \approx 25.53$ following [70] 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 $\varepsilon = 0.1$ we get a target $\delta_0 = 1.0046578$ by Lemma 5.12 and thus $m = 781$. Computing the expected number of clock cycles according to the various models available to us, we end up with the estimates shown in Table 5.

model	block size k	log clock cycles
enum	318	209
sieve	318	154

Table 5. Estimated clock cycles needed to mount a lattice reduction attack, parametrised as above, when using modulus switching.

6.3 Bai and Galbraith's embedding

Bai and Galbraith [14] show that for binary secret (that is, $\mathbf{s}_{(i)} \leftarrow \{0, 1\}$ or $\mathbf{s}_{(i)} \leftarrow \{-1, 0, 1\}$) we may embed our LWE lattice into a different lattice with uSVP structure than the one considered in Section 5.5. Let $m' = m + n$. The lattice is then $L = \{\mathbf{v} \in \mathbb{Z}^{m'} \mid \mathbf{A}'\mathbf{v} \equiv 0 \bmod q\}$ where $\mathbf{A}' = (\mathbf{A} \mid \mathbf{I}_m)$. This method is more efficient than the usual Kannan embedding lattice as discussed in Section 5.5.

The target short vector is now $(\mathbf{s} \parallel \mathbf{e})$ (as opposed to $\mathbf{e}$ as in the Kannan embedding case) which contains among its components those of $\mathbf{s}$. Bai and Galbraith observe that this enables us to take advantage of the smallness of $\mathbf{s}$ (see [14]). In particular, it is its smallness compared with the size of the error which is exploited. That is, where we have $\|\mathbf{s}\| \ll \|\mathbf{e}\|$, we may rescale the lattice into which the instance is embedded, increasing its volume. This increases the δ_0 which is required to solve the instance. In the case $\mathbf{s}_{(i)} \leftarrow \{-1, 0, 1\}$, after an appropriate rescaling, the volume of the lattice is increased by σ^n, where $\sigma = \alpha q/\sqrt{2\pi}$ is approximately the standard deviation of the error. In the case $\mathbf{s}_{(i)} \leftarrow \{0, 1\}$ the volume is increased by $(2\sigma)^n$ because we can scale by 2σ and then rebalance. We note that (in the terminology of Section 5.5) Bai and Galbraith use $t = 1$ rather than $t = \|\mathbf{e}\|$. In our lemma below, we adapted their theorem to $t = \|\mathbf{e}\|$. In our experiments we then use τ' to estimate the cost of lattice reduction for $t = 1$.

Lemma 6.4. *Let a small secret LWE instance be characterised by n, α, q, let $\mathbf{s}_{(i)} \leftarrow_{\$} \{a, \ldots, b\}$, let $\xi = 2/(b-a)$ and let $\sigma = \alpha q/\sqrt{2\pi}$. Any lattice reduction algorithm achieving log root-Hermite factor*

$$\log \delta = \frac{\left(\log(q/\sigma) - \log(2\tau\sqrt{\pi e})\right)^2 \cdot \log(q/\sigma)}{n\left(2\log(q/\sigma) - \log \xi\right)^2}$$

solves LWE by reducing BDD to uSVP for some fixed $\tau \leq 1$ if we have that

$$(q^m(\xi\sigma)^n)^{1/(m+n)}\sqrt{\frac{m+n}{2\pi e}} \leq q \tag{3}$$

where $m = m' - n = \sqrt{n(\log q - \log \sigma)/\log \delta} - n$.

Proof. We observe that scaling and rebalancing for a secret sampled from the interval $[a, \ldots, b]$ increases the volume by a factor of $(\xi\sigma)^n$. Hence, by [14, Section 6.2] and assumption (3) we have a gap of roughly

$$\frac{\lambda_2}{\lambda_1} = \frac{(q^m(\xi\sigma)^n)^{1/(m+n)}\sqrt{\frac{m+n}{2\pi e}}}{\sqrt{2m+n}\cdot\sigma}.$$

Following the notation of [14], let $m' = m + n$. By the same argument as in the discussion in Section 5.5 we have $\lambda_2/\lambda_1 \geq \tau\delta^{m'}$. Again for simplicity we assume equality. By [14, Lemma 1] the optimal value of m' is $m' = \sqrt{n(\log q - \log \sigma)/\log \delta}$. Therefore we have

$$\frac{\left(q^{m'-n}(\xi\sigma)^n\right)^{1/m'}}{\sqrt{4\pi e}\,\sigma} = \tau\,\delta^{m'}.$$

Taking logarithms and rearranging we get

$$\left(1 - \frac{n}{m'}\right)\log q + \frac{n}{m'}\log(\xi\sigma) = m'\log\delta + \log\sigma + \log\left(\tau\sqrt{4\pi e}\right).$$

Solving for $\log \delta_0$:

$$\log\delta_0 = \frac{m'\left(\log(q/\sigma) - \log\left(\tau\sqrt{4\pi e}\right)\right) + n\log\xi - n\log\left(q/\sigma\right)}{m'^2}$$

Substituting $m' = \sqrt{n(\log q - \log\sigma)/\log\delta}$:

$$\log\delta_0 = \frac{\sqrt{n(\log q - \log\sigma)/\log\delta}\left(\log(q/\sigma) - \log\left(\tau\sqrt{4\pi e}\right)\right) + n\log\xi - n\log\left(q/\sigma\right)}{\sqrt{n(\log q - \log\sigma)/\log\delta}^2}$$

$$1 = \frac{\sqrt{n(\log q - \log\sigma)/\log\delta}\left(\log(q/\sigma) - \log\left(\tau\sqrt{4\pi e}\right)\right) + n\log\xi - n\log\left(q/\sigma\right)}{n\log(q/\sigma)}.$$

Solving for $\sqrt{n(\log q - \log\sigma)/\log\delta}$:

$$\sqrt{\frac{n(\log q - \log\sigma)}{\log\delta}} = \frac{2n\log\left(q/\sigma\right) - n\log\xi}{\log\left(q/\sigma\right) - \log\left(\tau\sqrt{4\pi e}\right)}.$$

Finally, solving for $\log\delta_0$:

$$\log\delta = \frac{\left(\log(q/\sigma) - \log(\tau\sqrt{4\pi e})\right)^2 \cdot \log(q/\sigma)}{n\left(2\log(q/\sigma) - \log\xi\right)^2}.$$

□

Remark 6.5. Specialising Lemma 6.4 to $\mathbf{s}_{(i)} \hookleftarrow_{\$} \{-1, 0, 1\}$ and hence $\xi = 1$ gives

$$\log \delta = \frac{(\log \alpha\tau\sqrt{2e})^2}{4n(\log q - \log \frac{\alpha q}{\sqrt{2\pi}})}.$$

Bai and Galbraith also observe that, perhaps counterintuitively, modulus switching does not improve their algorithm. This is because modulus switching results in a smaller rescaling factor and hence leaves a smaller gap.

6.4 Small secret BKW

In this section we consider the small secret variant of BKW described in [8]. 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 6.6 ([8]). *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 $\mathcal{O}(2^{cn} \cdot n \log^2 n)$. Here the complexity of solving is

$$\mathcal{O}\left(2^{n(c + \frac{\log d}{\log n})} \cdot n \log^2 n\right),$$

using naive modulus switching. Using lazy modulus switching [8, Corollary 3], 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.

6.5 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 estimate the cost of running the algorithms discussed in Sections 5 and 6 for parameter sets from the literature. Our estimator is available at [4]. We consider the following parameter sets.

- *Regev*: These are Regev's example choices for parameters from [70]. We use [9] to pick $q \approx n^2$ and $\alpha = 1/(\sqrt{2\pi n}\log_2^2 n)$.
- *LindnerPeikert*: We use [9] to select parameters as suggested in [52] 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 $\alpha = 3.2\sqrt{2\pi}/q$ inspired by parameters suggested in [36]. We always assume $\mathbf{s}_{(i)} \leftarrow_{\$} \{0,1\}$, which means our secret is a bit bigger than in [36] where the secret has Hamming weight $\lambda/2$ regardless of dimension.

In Tables 6–11, "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, "DEC" to the algorithm discussed in Section 5.4, "Kannan" to the algorithm discussed in Section 5.5, "Bai-Gal" to the algorithm discussed in Section 6.3 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 "bop" 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 "$L_{\mathbf{s},\chi}$" refer to the number of calls to the LWE oracle. The columns "bkz2" resp. "sieve" refer to BKZ 2.0 estimates based on the row "enum" resp. "sieve" in Table 1. We use the "bkz2" estimates to optimise parameters for lattice-based algorithms. The column "enum" gives the number of enumerations in the decoding stage of "DEC". The column "g" is the number of components that are guessed before running the respective algorithms as discussed in Section 6.1 (this only applies to small secret instances). All columns list the logarithm to base two of their respective values.

In all cases, costs are overall, i.e. we give an estimate for the overall cost of solving, including repeated trials and repeated guesses. If "–" is given 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. We always use $\tau' = 0.3$ when considering Kannan embedding or the embedding by Bai and Galbraith. For each choice of parameter set and for each n, we highlight the entry giving the runtime of the algorithm which runs fastest in that case.

8 Discussion

The problems of giving the concrete hardness of the LWE problem are manifold.

No closed formulae. For most algorithms, there is no sufficiently precise closed formula which expresses the running time in terms of the parameters specifying the problem (e.g., n, q, α), mainly due to a lack of a closed formula for lattice reduction as a function of δ_0. 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. For example, by our estimates the parameter choices made in [36] are too conservative, as first observed by van de Pol and Smart [77], even in light of specialised algorithms exploiting the presence of a small secret (cf. Section 6.3). 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.

	BKW			SIS			DEC			Kannan		
n	bop	mem	$L_{s,\chi}$	bkz2	sieve	$L_{s,\chi}$	bop	enum	$L_{s,\chi}$	bkz2	sieve	$L_{s,\chi}$
64	57	50	45	**34**	50	12	**34**	18	10	35	50	13
128	101	93	87	75	86	27	65	49	11	**63**	73	14
256	189	181	174	205	171	52	168	152	45	207	**153**	15
512	366	358	350	566	339	81	453	437	46	665	**335**	16
1024	**721**	712	703	1478	732	206	1203	1187	111	1962	747	17

Table 6. Regev.

	BKW			SIS			DEC			Kannan			Arora-GB		
n	bop	mem	$L_{s,\chi}$	bkz2	sieve	$L_{s,\chi}$	bop	enum	$L_{s,\chi}$	bkz2	sieve	$L_{s,\chi}$	bop	mem	$L_{s,\chi}$
64	54	47	42	**34**	49	12	**34**	18	10	35	50	13	289	286	92
128	89	82	76	70	81	22	60	44	11	**57**	69	14	505	501	389
256	160	151	146	179	156	47	146	130	44	181	**140**	15	—	—	—
512	**289**	281	273	476	308	86	376	361	45	579	303	16	—	—	—
1024	**548**	541	532	1190	636	201	954	938	110	1643	652	17	—	—	—

Table 7. LindnerPeikert.

	MitM			BKW			SIS				DEC				Kannan				Bai-Gal			
n	bop	mem	$L_{s,\chi}$	bop	mem	$L_{s,\chi}$	bkz2	sieve	$L_{s,\chi}$	g	bop	enum	$L_{s,\chi}$	g	bkz2	sieve	$L_{s,\chi}$	g	bkz2	sieve	$L_{s,\chi}$	g
64	43	32	6	46	39	34	**34**	49	11	0	**34**	18	10	0	35	50	13	0	**34**	49	11	0
128	76	64	7	76	68	62	63	76	22	0	54	39	11	0	51	70	14	0	**37**	52	12	0
256	141	128	8	138	130	123	162	146	46	3	135	119	44	0	155	146	55	40	**76**	83	13	0
512	270	256	9	249	239	233	403	368	248	200	345	329	45	0	397	384	287	272	239	**168**	14	0
1024	527	512	10	**481**	473	464	900	862	738	688	838	822	525	480	895	870	759	744	700	519	287	272

Table 8. Regev with $s_{(i)} \leftarrow_{\$} \{0, 1\}$.

	MitM			BKW			SIS				DEC				Kannan				Bai-Gal			
n	bop	mem	$L_{s,\chi}$	bop	mem	$L_{s,\chi}$	bkz2	sieve	$L_{s,\chi}$	g	bop	enum	$L_{s,\chi}$	g	bkz2	sieve	$L_{s,\chi}$	g	bkz2	sieve	$L_{s,\chi}$	g
64	42	32	6	46	39	34	**34**	49	11	0	**34**	18	10	0	35	50	13	0	**34**	49	11	0
128	75	64	7	75	68	62	75	85	26	0	60	44	19	0	62	73	14	0	**37**	52	12	0
256	141	128	8	136	129	122	191	185	97	60	157	141	44	0	181	184	110	96	**90**	91	13	0
512	270	256	9	255	248	240	450	449	369	336	413	397	237	192	439	441	366	352	307	**233**	78	64
1024	527	512	10	**506**	464	489	952	946	857	824	913	897	733	688	942	942	862	848	788	687	510	496

Table 9. LindnerPeikert with $\mathbf{s}_{(i)} \leftarrow_{\$} \{0, 1\}$.

	MitM			BKW			SIS				DEC				Kannan				Bai-Gal			
n	bop	mem	$L_{s,\chi}$	bop	mem	$L_{s,\chi}$	bkz2	sieve	$L_{s,\chi}$	g	bop	enum	$L_{s,\chi}$	g	bkz2	sieve	$L_{s,\chi}$	g	bkz2	sieve	$L_{s,\chi}$	g
64	45	32	6	53	42	37	**31**	46	9	0	32	16	10	0	34	49	13	0	34	49	11	0
128	78	64	7	86	74	68	**34**	49	10	0	36	20	11	0	37	52	14	0	37	52	12	0
256	143	128	8	148	136	129	**37**	52	11	0	39	23	12	0	40	56	15	0	40	55	13	0
512	272	256	9	272	260	252	**40**	55	12	0	42	26	13	0	44	59	16	0	43	58	14	0
1024	529	512	10	520	507	498	69	79	13	0	66	50	14	0	67	78	17	0	**59**	78	15	0
2048	1042	1024	11	1015	1003	993	214	162	20	0	200	184	15	0	205	156	18	0	173	**141**	16	0

Table 10. FHE with $L = 2$ with $\mathbf{s}_{(i)} \leftarrow_{\$} \{0, 1\}$.

	MitM			BKW			SIS				DEC				Kannan				Bai-Gal			
n	bop	mem	$L_{s,\chi}$	bop	mem	$L_{s,\chi}$	bkz2	sieve	$L_{s,\chi}$	g	bop	enum	$L_{s,\chi}$	g	bkz2	sieve	$L_{s,\chi}$	g	bkz2	sieve	$L_{s,\chi}$	g
512	274	256	9	287	270	262	**40**	55	12	0	43	28	13	0	43	58	15	0	43	58	14	0
1024	531	512	10	540	523	514	**43**	58	13	0	46	30	14	0	47	62	16	0	46	62	15	0
2048	1044	1024	11	—	—	—	**46**	61	14	0	50	35	15	0	50	65	17	0	50	65	16	0
4096	2069	2048	12	—	—	—	**49**	64	15	0	52	36	16	0	53	68	18	0	53	68	17	0
8192	4118	4096	13	—	—	—	105	106	16	0	113	98	17	0	107	108	19	0	**104**	106	18	0
16384	8215	8192	14	—	—	—	328	213	17	0	332	316	18	0	326	214	20	0	315	**210**	19	0

Table 11. FHE with $L = 10$ with $\mathbf{s}_{(i)} \leftarrow_{\$} \{0, 1\}$.

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, DEC may be favourable (see, e.g., Table 7). For large n, BKW may be fastest when considering public-key encryption (see, e.g., Table 6) but not when considering homomorphic encryption schemes which require large q (cf. Tables 10 and 11).

We note that while the Arora–Ge algorithm and its Gröbner basis variants always perform much worse than other algorithms in our tests, it is shown in [7] that this family of algorithms outperforms other families when considering a particular variant of LWE, i.e. UniformNoise-LWE instances.

Finally, we note that all families of algorithms discussed in this work permit parallelisation. For BKW we can distribute (partial) elimination tables across computing units and for the Arora-GB algorithm we can rely on parallelised linear algebra routines. Lattice reduction, too, can be easily distributed for the instances considered in this work, as we are running in the low advantage regime, i.e. we are computing many independent lattice reductions on fresh or re-randomised inputs.

Time-memory trade-offs. According to our estimates of running BKZ, using sieving as the SVP oracle is faster than enumeration for large n. While this is to be expected given that sieving is asymptotically faster than enumeration, it might be surprising to see the crossover already for dimension $n = 256$ in some cases. It is important to note, however, that sieving would require an amount of memory so substantial that, for most parameters we consider, it is not clear that sieving is worth considering even where it is faster "on paper". A completely analogous statement can be made when considering the BKW algorithm or the Arora–Ge and its Gröbner basis variant. Indeed, all algorithms which achieve a time complexity of $2^{\mathcal{O}(n)}$ also require memory of the order of $2^{\mathcal{O}(n)}$. An interesting open question is hence if an algorithm exists which solves LWE in $2^{\mathcal{O}(n)}$ operations but requiring only $\mathrm{poly}(n)$ memory.

Incomplete data. For, say, the decoding approach we are able to trade running time for success probability. On the other hand, we do not know how to do this when using Kannan embedding. As highlighted in Section 5.5, for a success probability of 0.1 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. While the decoding approach seems to outperform the application of Kannan embedding as highlighted in Example 5.20, by a similar argument, it is to be expected that the algorithm of Bai and Galbraith could be shown to produce better results if such data was available.

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 allow to refine these 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.

Acknowledgement: We thank Steven Galbraith, Paul Kirchner and Cong Ling for pointing out mistakes and oversights in an earlier version of this work.

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

References

[1] D. Aggarwal, D. Dadush, O. Regev and N. Stephens-Davidowitz, Solving the shortest vector problem in 2^n time via discrete gaussian sampling, preprint (2014), http://arxiv.org/abs/1412.7994.

[2] M. Ajtai, Generating hard instances of lattice problems (extended abstract), in: *Theory of Computing* (STOC 1996), ACM Press, New York (1996), 99–108.

[3] M. Albrecht, BKW-LWE, 2013, https://bitbucket.org/malb/bkw-lwe.
[4] M. Albrecht, Estimator for the bit security of LWE instances, 2015, https://bitbucket.org/malb/lwe-estimator.
[5] M. Albrecht, D. Cadé, X. Pujol and D. Stehlé, fplll, development version, 2014, https://github.com/dstehle/fplll.
[6] M. R. Albrecht, C. Cid, J.-C. Faugère, R. Fitzpatrick and L. Perret, On the complexity of the BKW algorithm on LWE, *Des. Codes Cryptogr.* **74** (2015), 325–354.
[7] M. R. Albrecht, C. Cid, J.-C. Faugère and L. Perret, Algebraic algorithms for LWE, preprint (2014), http://eprint.iacr.org/2014/1018.
[8] M. R. Albrecht, J.-C. Faugère, R. Fitzpatrick and L. Perret, Lazy modulus switching for the BKW algorithm on LWE, in: *Public-Key Cryptography* (PKC 2014) Lecture Notes in Comput. Sci. 8383, Springer, Berlin (2014), 429–445.
[9] M. R. Albrecht, R. Fitzpatrick, D. Cabracas, F. Göpfert and M. Schneider, A generator for LWE and Ring-LWE instances, preprint (2013), www.iacr.org/news/files/2013-04-29lwe-generator.pdf.
[10] M. R. Albrecht, R. Fitzpatrick and F. Göpfert, On the efficacy of solving LWE by reduction to unique-SVP, in: *Information Security and Cryptology* (ICISC 13), Lecture Notes in Comput. Sci. 8565, Springer, Cham (2014), 293–310.
[11] B. Applebaum, D. Cash, C. Peikert and A. Sahai, Fast cryptographic primitives and circular-secure encryption based on hard learning problems, in: [40], 595–618.
[12] S. Arora and R. Ge, New algorithms for learning in presence of errors, in: *Automata, Languages and Programming* (ICALP 2011), part I, Lecture Notes in Comput. Sci. 6755, Springer, Berlin (2011), 403–415.
[13] L. Babai, On Lovász' lattice reduction and the nearest lattice point problem (shortened version), in: *Theoretical Aspects of Computer Science* (STACS 1986), Lecture Notes in Comput. Sci 182, Springer, Berlin (1985), 13–20.
[14] S. Bai and S. D. Galbraith, Lattice decoding attacks on binary LWE, in: *Information Security and Privacy* (ACISP 2014), Lecture Notes in Comput. Sci. 8544, Springer, Berlin (2014), 322–337.
[15] D. J. Bernstein, J. Buchmann and E. Dahmen (eds.), *Post-Quantum Cryptography*, Springer, Berlin, 2009.
[16] A. Blum, A. Kalai and H. Wasserman, Noise-tolerant learning, the parity problem, and the statistical query model, *J. ACM* **50** (2003), no. 4, 506–519.
[17] Z. Brakerski, C. Gentry and V. Vaikuntanathan, (Leveled) fully homomorphic encryption without bootstrapping, in: *Innovations in Theoretical Computer Science* (ITCS 2012), ACM Press, New York (2012), 309–325.
[18] Z. Brakerski, A. Langlois, C. Peikert, O. Regev and D. Stehlé, Classical hardness of learning with errors, in: *Theory of Computing* (STOC 2013), ACM Press, New York (2013), 575–584.
[19] Z. Brakerski and V. Vaikuntanathan, Efficient fully homomorphic encryption from (standard) LWE, in: *Foundations of Computer Science* (FOCS 2011), IEEE Computer Society Press, Los Alamitos (2011), 97–106.
[20] P. Bürgisser, M. Clausen and M. A. Shokrollahi, *Algebraic Complexity Theory*, Grundlehren Math. Wiss. 315, Springer, Berlin, 1997.
[21] D. Cadé, X. Pujol and D. Stehlé, fpLLL 4.0.4, 2013, http://perso.ens-lyon.fr/damien.stehle/fplll/.
[22] D. Cash, D. Hofheinz, E. Kiltz and C. Peikert, Bonsai trees, or how to delegate a lattice basis, *J. Cryptology* **25** (2012), no. 4, 601–639.
[23] Y. Chen, *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*, Ph.D. thesis, Paris 7, 2013.
[24] Y. Chen and P. Q. Nguyen, BKZ 2.0: Better lattice security estimates, in: *Advances in Cryptology* (Asiacrypt 2011), Lecture Notes in Comput. Sci. 7073, Springer, Berlin (2011), 1–20.
[25] Y. Chen and P. Q. Nguyen, BKZ 2.0: Better lattice security estimates (full version), preprint (2012), www.di.ens.fr/~ychen/research/Full_BKZ.pdf.
[26] H. Chernoff, A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations, *Ann. Math. Stat.* **23** (1952), 493–507.
[27] A. Duc, F. Tramèr and S. Vaudenay, Better algorithms for LWE and LWR, preprint (2015), http://eprint.iacr.org/2015/056.
[28] L. Ducas-Binda, *Signatures fondées sur les réseaux euclidiens: attaques, analyses et optimisations*, Ph.D. thesis, École Normale Supérieure Paris, 2013.
[29] N. Gama and P. Q. Nguyen, Predicting lattice reduction, in: *Advances in Cryptology* (Eurocrypt 2008), Lecture Notes in Comput. Sci. 4965, Springer, Berlin (2008), 31–51.
[30] N. Gama, P. Q. Nguyen and O. Regev, Lattice enumeration using extreme pruning, in: [38], 257–278.
[31] S. Garg, C. Gentry and S. Halevi, Candidate multilinear maps from ideal lattices, in: *Advances in Cryptology* (Eurocrypt 2013), Lecture Notes in Comput. Sci. 7881, Springer, Berlin (2013), 1–17.
[32] C. Gentry, *A fully homomorphic encryption scheme*, Ph.D. thesis, Stanford University, 2009.
[33] C. Gentry, S. Gorbunov and S. Halevi, Graph-induced multilinear maps from lattices, preprint (2014), http://eprint.iacr.org/2014/645.
[34] C. Gentry, S. Halevi and N. P. Smart, Fully homomorphic encryption with polylog overhead, in: *Advances in Cryptology* (Eurocrypt 2012), Lecture Notes in Comput. Sci. 7237, Springer, Berlin (2012), 465–482.
[35] C. Gentry, S. Halevi and N. P. Smart, Homomorphic evaluation of the AES circuit, in: *Advances in Cryptology* (Crypto 2012), Lecture Notes in Comput. Sci. 7417, Springer, Berlin (2012), 850–867.
[36] C. Gentry, S. Halevi and N. P. Smart, Homomorphic evaluation of the AES circuit, preprint (2012), http://eprint.iacr.org/2012/099.

[37] C. Gentry, C. Peikert and V. Vaikuntanathan, Trapdoors for hard lattices and new cryptographic constructions, in: *Theory of Computing* (STOC 2008), ACM Press, New York (2008), 197–206.
[38] H. Gilbert (ed.), *Advances in Cryptology* (Eurocrypt 2010), Lecture Notes in Comput. Sci. 6110, Springer, Berlin, 2010.
[39] S. Goldwasser, Y. Kalai, C. Peikert and V. Vaikuntanathan, Robustness of the learning with errors assumption, in: *Innovations in Computer Science* (ICS 2010), Tsinghua University Press, Beijing (2010), 230–240.
[40] S. Halevi (ed.), *Advances in Cryptology* (Crypto 2009), Lecture Notes in Comput. Sci. 5677, Springer, Berlin, 2009.
[41] G. Hanrot, X. Pujol and D. Stehlé, Algorithms for the shortest and closest lattice vector problems, in: *Coding and Cryptology*, Lecture Notes in Comput. Sci. 6639, Springer, Berlin (2011), 159–190.
[42] G. Hanrot, X. Pujol and D. Stehlé, Analyzing blockwise lattice algorithms using dynamical systems, in: *Advances in Cryptology* (Crypto 2011), Lecture Notes in Comput. Sci. 6841, Springer, Berlin (2011), 447–464.
[43] W. Hart, F. Johansson and S. Pancratz, FLINT: Fast Library for Number Theory, 2014. Version 2.4.4, http://flintlib.org.
[44] E. Jones et al., SciPy: Open source scientific tools for Python, 2001, www.scipy.org.
[45] A. Joux, *Algorithmic Cryptanalysis*, CRC Press, Boca Raton, 2009.
[46] R. Kannan, Minkowski's convex body theorem and integer programming, *Math. Oper. Res.* **12** (1987), no. 3, 415–440.
[47] M. J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire and L. Sellie, On the learnability of discrete distributions, in: *Theory of Computing* (STOC 1994), ACM Press, New York (1994), 273–282.
[48] T. Laarhoven, Sieving for shortest vectors in lattices using angular locality-sensitive hashing, preprint (2014), http://eprint.iacr.org/2014/744.
[49] T. Laarhoven and B. de Weger, Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing, preprint (2015), http://eprint.iacr.org/2015/211.
[50] A. Lenstra, J. Lenstra, H.W. and L. Lovász, Factoring polynomials with rational coefficients, *Math. Ann.* **261** (1982), no. 4, 515–534.
[51] T. Lepoint and M. Naehrig, A comparison of the homomorphic encryption schemes FV and YASHE, in: *Progress in Cryptology* (Africacrypt 2014), Lecture Notes in Comput. Sci. 8469, Springer, Berlin (2014), 318–335.
[52] R. Lindner and C. Peikert, Better key sizes (and attacks) for LWE-based encryption, in: *Topics in Cryptology* (CT-RSA 2011), Lecture Notes in Comput. Sci. 6558, Springer, Berlin (2011), 319–339.
[53] C. Ling, S. Liu, L. Luzzi and D. Stehlé, Decoding by embedding: Correct decoding radius and DMT optimality, in: *Information Theory Proceedings* (ISIT 2011), IEEE Computer Society, Los Alamitos (2011), 1106–1110.
[54] M. Liu and P. Q. Nguyen, Solving BDD by enumeration: An update, in: *Topics in Cryptology* (CT-RSA 2013), Lecture Notes in Comput. Sci. 7779, Springer, Berlin (2013), 293–309.
[55] L. Lovász, *An Algorithmic Theory of Numbers, Graphs and Convexity*, CBMS-NSF Regional Conf. Ser. in Appl. Math., SIAM, Philadelphia, 1986.
[56] V. Lyubashevsky and D. Micciancio, On bounded distance decoding, unique shortest vectors, and the minimum distance problem, in: [40], 577–594.
[57] D. Micciancio and C. Peikert, Hardness of SIS and LWE with small parameters, in: *Advances in Cryptology* (Crypto 2013), part I, Lecture Notes in Comput. Sci. 8042, Springer, Berlin (2013), 21–39.
[58] D. Micciancio and O. Regev, Lattice-based cryptography, in: [15], 147–191.
[59] D. Micciancio and M. Walter, Fast lattice point enumeration with minimal overhead, in: *Discrete Algorithms* (SODA 2015), ACM-SIAM (2015), 276–294.
[60] J. J. Moré, B. S. Garbow and K. E. Hillstrom, User guide for MINPACK-1, ANL-80-74, Argonne National Laboratory, 1980, www.mcs.anl.gov/~more/ANL8074a.pdf.
[61] P. Q. Nguyen, Hermite's constant and lattice algorithms, in: [64], 19–69.
[62] P. Q. Nguyen, Lattice reduction algorithms: Theory and practice (invited talk), in: *Advances in Cryptology* (Eurocrypt 2011), Lecture Notes in Comput. Sci. 6632, Springer, Berlin (2011), 2–6.
[63] P. Q. Nguyen and D. Stehlé, Floating-point LLL revisited, in: *Advances in Cryptology* (Eurocrypt 2005), Lecture Notes in Comput. Sci. 3494, Springer, Berlin (2005), 215–233.
[64] P. Q. Nguyen and B. Vallée (eds.), *The LLL Algorithm: Survey and Applications*, Inf. Secur. Cryptography, Springer, Dordrecht, 2010.
[65] C. Peikert, Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract, in: *Theory of Computing* (STOC 2009), ACM Press, New York (2009), 333–342.
[66] C. Peikert, V. Vaikuntanathan and B. Waters, A framework for efficient and composable oblivious transfer, in: *Advances in Cryptology* (Crypto 2008), Lecture Notes in Comput. Sci. 5157, Springer, Berlin (2008), 554–571.
[67] C. Peikert and B. Waters, Lossy trapdoor functions and their applications, *SIAM J. Comput.* **40** (2011), no. 6, 1803–1844.
[68] C. Pernet, *High performance and reliable algebraic computing*, Habilitation à Diriger des Recherches in Symbolic Computation, Université Joseph Fourier, Grenoble, 2014.
[69] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, in: *Theory of Computing* (STOC 2005), ACM Press, New York (2005), 84–93.
[70] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, *J. ACM* **56** (2009), Article No. 34.
[71] O. Regev, The learning with errors problem (invited survey), in: *IEEE Conference on Computational Complexity*, IEEE Computer Society, Los Alamitos (2010), 191–204.

[72] C. P. Schnorr, Lattice reduction by random sampling and birthday methods, in: *Theoretical Aspects on Computer Science* (STACS 2003), Lecture Notes in Comput. Sci. 2607, Springer, Berlin (2003), 145–156.
[73] C. P. Schnorr and M. Euchner, Lattice basis reduction: Improved practical algorithms and solving subset sum problems, *Math. Program.* **66** (1994), 181–199.
[74] V. Shoup, Number Theory Library 5.5.2 (NTL) for C++, www.shoup.net/ntl/.
[75] D. Stehlé, An overview of lattice reduction algorithms, Invited talk at ICISC, 2013.
[76] W. Stein et al., Sage Mathematics Software Version 6.3, The Sage Development Team, 2014, www.sagemath.org.
[77] J. van de Pol and N. P. Smart, Estimating key sizes for high dimensional lattice-based systems, in: *Cryptography and Coding* (IMACC 2013), Lecture Notes in Comput. Sci. 8308, Springer, Berlin (2013), 290–303.
[78] M. van Dijk, C. Gentry, S. Halevi and V. Vaikuntanathan, Fully homomorphic encryption over the integers, in: [38], 24–43.
[79] M. Walter, Lattice point enumeration on block reduced bases, preprint (2014), http://eprint.iacr.org/2014/948.

Received March 19, 2015; revised September 20, 2015; accepted September 24, 2015.