On the complexity of the BKW algorithm on LWE

Martin Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, Ludovic Perret

On the Complexity of the BKW Algorithm on LWE

Martin R. Albrecht[1], Carlos Cid[3], Jean-Charles Faugère[2], Robert Fitzpatrick[3], and Ludovic Perret[2]

[1] Technical University of Denmark, Denmark
[2] INRIA, Paris-Rocquencourt Center, POLSYS Project
UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
CNRS, UMR 7606, LIP6, F-75005, Paris, France
[3] Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, United Kingdom
maroa@dtu.dk, carlos.cid@rhul.ac.uk, jean-charles.faugere@inria.fr,
robert.fitzpatrick.2010@live.rhul.ac.uk, ludovic.perret@lip6.fr

Abstract. This work presents a study of the complexity of the Blum-Kalai-Wasserman (BKW) algorithm when applied to the Learning with Errors (LWE) problem, by providing refined estimates for the data and computational effort requirements for solving *concrete* instances of the LWE problem. We apply this refined analysis to suggested parameters for various LWE-based cryptographic schemes from the literature and compare with alternative approaches based on lattice reduction. As a result, we provide new upper bounds for the concrete hardness of these LWE-based schemes. Rather surprisingly, it appears that BKW algorithm outperforms known estimates for lattice reduction algorithms starting in dimension $n \approx 250$ when LWE is reduced to SIS. However, this assumes access to an unbounded number of LWE samples.

1 Introduction

LWE (Learning with Errors) is a generalisation for large moduli of the well-known LPN (Learning Parity with Noise) problem. It was introduced by Regev in [29] and has provided cryptographers with a remarkably flexible tool for building cryptosystems. For example, Gentry, Peikert and Vaikuntanathan presented in [17] LWE-based constructions of trapdoor functions and identity-based encryption. Moreover, in his recent seminal work Gentry [16] resolved one of the longest standing open problems in cryptography with a construction related to LWE: the first fully homomorphic encryption scheme. This was followed by further constructions of (fully) homomorphic encryption schemes based on the LWE problem, e.g. [5, 11]. Reasons for the popularity of LWE as a cryptographic primitive include its simplicity as well as convincing theoretical arguments regarding its hardness, namely, a reduction from worst-case lattice problems, such as the (decision) Shortest Vector Problem (GapSVP) and Short Independent Vectors Problem (SIVP), to average-case LWE [29, 10].

Definition 1 (LWE [29]). *Let n, q be positive integers, χ be a probability distribution on $\mathbb{Z}$ and $\mathbf{s}$ be a secret vector following the uniform distribution on $\mathbb{Z}_q^n$. We denote by $L_{\mathbf{s},\chi}$ the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a}$ from the uniform distribution on $\mathbb{Z}_q^n$, choosing $e \in \mathbb{Z}$ according to χ, and returning $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.*
– Search-LWE *is the problem of finding $\mathbf{s} \in \mathbb{Z}_q^n$ given pairs $(\mathbf{a}_i, c_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ sampled according to $L_{\mathbf{s},\chi}$.*
– Decision-LWE *is the problem of deciding whether pairs $(\mathbf{a}_i, c_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ are sampled according to $L_{\mathbf{s},\chi}$ or the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.*

The modulus q is typically taken to be polynomial in n, and χ is the discrete Gaussian distribution on $\mathbb{Z}$ with mean 0 and standard deviation $\sigma = \alpha \cdot q$, for some α.[1] For these choices it was shown in [29, 10] that if $\sqrt{2\pi}\sigma > 2\sqrt{n}$, then (worst-case) GapSVP $- \tilde{\mathcal{O}}(n/\alpha)$ reduces to (average-case) LWE.

Motivation. While there is a reduction of LWE to (assumed) hard lattice problems [29], little is known about the *concrete* hardness of particular LWE instances. That is, given particular values for σ, q and n, what is the computational cost to recover the secret using currently known algorithms? As a consequence of this gap, most proposals based on LWE do not provide concrete choices for parameters and restrict themselves to asymptotic statements about security, which can be considered unsatisfactorily vague for practical purposes. In fact we see this lack of precision as one of the several obstacles to the consideration of LWE-based schemes for real-world applications.

Previous Work. We may classify algorithms for solving LWE into two families. The first family reduces LWE to the problem of finding a short vector in the (scaled) dual lattice (commonly known as the Short Integer Solution (SIS) problem) constructed from a set of LWE samples. The second family solves the Bounded Distance Decoding (BDD) problem in the primal lattice. For both families lattice reduction algorithms may be applied. We may either use lattice reduction to find a short vector in the dual lattice or apply lattice reduction and (a variant of) Babai's algorithm to solve BDD [21]. Indeed, the expected complexity of lattice algorithms is often exclusively considered when parameters for LWE-based schemes are discussed. However, while the effort on improving lattices algorithms is intense [31, 12, 26, 15, 27, 19, 25, 28], our understanding of the behaviour of these algorithms in high dimensions is still limited.

On the other hand, combinatorial algorithms for tackling the LWE problem remain rarely investigated from an algorithmic point of view. For example, the main subject of this paper – the BKW algorithm – specifically applied to the LWE problem has so far received no treatment in the literature[2]. However, since the BKW algorithm can be viewed as an oracle producing short vectors in the dual lattice spanned by the $\mathbf{a}_i$ (i.e., it reduces LWE to SIS) it shares some similarities with combinatorial (exact) SVP solvers. Finally, recently a new algorithm for LWE that reduces the problem to BDD but does not make calls to lattice reduction algorithms has been proposed: Arora and Ge [7] proposed a new algebraic technique for solving LWE. The algorithm has a total complexity (time and space) of $2^{\tilde{\mathcal{O}}(\sigma^2)}$ and is thus subexponential when $\sigma \leq \sqrt{n}$, remaining exponential when $\sigma > \sqrt{n}$. It is worth noting that Arora and Ge achieve the $\sqrt{n}$ hardness-threshold found by Regev [29], and thus provide a subexponential algorithm precisely in the region where the reduction to GapSVP fails. We note however that currently the main relevance of Arora-Ge's algorithm is asymptotic as the constants hidden in $\tilde{\mathcal{O}}(\cdot)$ are rather large [3]; it is an open question whether one can improve its practical efficiency.

Contribution. Firstly, we present a detailed study of a dedicated version of the Blum, Kalai and Wasserman (BKW) algorithm [9] for LWE with discrete Gaussian noise. The BKW algorithm is known to have (time and space) complexity $2^{\mathcal{O}(n)}$ when applied to LWE instances with a prime modulus polynomial in n [29]; in this paper we provide both the leading constant of the exponent in $2^{\mathcal{O}(n)}$ and concrete costs of BKW when applied to Search- and Decision-LWE. That is, by studying in detail all steps of the BKW algorithm, we 'de-asymptotic-ify' the understanding of the hardness of LWE under the BKW algorithm and provide concrete values for the expected number of operations for solving instances of the LWE problem. More precisely, we show the following theorem in Section 3.4.

[1] It is common in the literature on LWE to parameterise discrete Gaussian distributions by $s = \sigma\sqrt{2\pi}$ instead of σ. Since we are mainly interested in the "size" of the noise, we deviate from this standard in this work.

[2] However, a detailed study of the algorithm to the LPN case was provided [14], which in fact heavily inspired this work. The authors of [14] conducted a detailed analysis of the BKW algorithm as applied to LPN, while also giving revised security estimates for some HB-type authentication protocols relying on the hardness of LPN.

Theorem 1 (Search-LWE, simplified). *Let $(\mathbf{a}_i, c_i)$ be samples following $L_{\mathbf{s},\chi}$, set $a = \lfloor \log_2(1/(2\alpha)^2) \rceil$, $b = n/a$ and q a prime. Let d be a small constant $0 < d < \log_2(n)$. Assume α is such that $q^b = q^{n/a} = q^{n/\lfloor \log_2(1/(2\alpha)^2) \rceil}$ is superpolynomial in n. Then, given these parameters, the cost of the BKW algorithm to solve* Search-LWE *is*

$$\left(\frac{q^b-1}{2}\right) \cdot \left(\frac{a(a-1)}{2} \cdot (n+1)\right) + \left\lceil \frac{q^b}{2} \right\rceil \cdot \left(\left\lceil \frac{n}{d} \right\rceil + 1\right) \cdot d \cdot a + \mathrm{poly}(n) \approx \left(a^2 n\right) \cdot \frac{q^b}{2}$$

operations in $\mathbb{Z}_q$. Furthermore,

$$a \cdot \left\lceil \frac{q^b}{2} \right\rceil + \mathrm{poly}(n) \text{ calls to } L_{\mathbf{s},\chi} \text{ and storage of } \left(a \cdot \left\lceil \frac{q^b}{2} \right\rceil \cdot n\right) \text{ elements in } \mathbb{Z}_q \text{ are needed.}$$

We note that the above result is a corollary to our main theorem (Theorem 2) which depends on a value m. However, since at present no closed form expressing m is known, the above simplified statement avoids m by restricting choices on parameters of the algorithm. We also show the following simple corollary on the algorithmic hardness of Decision-LWE.

Corollary 1 (Decision-LWE). *Let $(\mathbf{a}_i, c_i)$ be samples following $L_{\mathbf{s},\chi}$, $0 < b \leq n$ be a parameter, $0 < \epsilon < 1$ the targeted success rate and $a = \lceil n/b \rceil$ the addition depth. Then, the expected cost of the BKW algorithm to distinguish $L_{\mathbf{s},\chi}$ from random with success probability ϵ is*

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

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

$$m \cdot \left(\frac{a}{2} \cdot (n+2)\right) \text{ with } m = \epsilon / \exp\left(-\frac{\pi^2 \sigma^2 2^{a+1}}{q^2}\right)$$

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.

This corollary is perhaps the more useful result for cryptographic applications which rely on Decision-LWE and do not assume a prime modulus q. Here, we investigate the search variant first because the decision variant follows easily. However, we emphasize that there are noticeable differences in the computational costs of the two variants. A reader only interested in Decision-LWE is invited to skip Sections 3.2 and 3.3.

In Section 4, we apply the BKW algorithm to various parameter choices for LWE from the literature [29, 21, 5] and compare with alternative approaches in Section 5. It appears that the BKW algorithm outperforms known estimates for lattice reduction algorithms when LWE is reduced to SIS (called "Distinguishing" in [21]) starting in dimension $n \approx 250$ (but, assuming access to an unbounded number of LWE samples). However, reducing LWE to BDD (called "Decoding" in [21]) and applying a combination of lattice reduction and decoding outperforms BKW for the parameter sets considered in this work. However, since the concrete behaviour of lattice reduction algorithms is not fully understood, the commonly used running-time estimates tend to be optimistic. In contrast, for combinatorial algorithms such as BKW, we have a much better understanding of the concrete complexity, leading to greater confidence in the recovered bounds. Finally, we report experimental results for small instances of LWE in Section 6.

2 Preliminaries

GAUSSIANS. Let $\mathcal{N}(\mu, \sigma^2)$ denote the Gaussian distribution with mean μ and standard deviation σ. The LWE problem considers a discrete Gaussian distribution over $\mathbb{Z}$ which is then reduced modulo q. This distribution in $\mathbb{Z}_q$ can be obtained by *discretising* the corresponding *wrapped* Gaussian distribution over the reals. To wrap $\mathcal{N}(0, \sigma^2)$ mod q, we denote by $p(\phi)$ the probability density function determined by σ, define the periodic variable $\theta := \phi \bmod q$ and let

$$p'(\theta) = \sum_{k=-\infty}^{\infty} p(\theta + qk) \text{ for } -q/2 < \theta \leq q/2. \tag{1}$$

As $|k|$ increases, the contribution of $p(\theta + qk)$ falls rapidly; in fact, exponentially fast [13]. Hence, we can pick a point at which we 'cut' $p'(\theta)$ and work with this approximation. We denote the distribution sampled according to p' and rounded to the nearest integer in the interval $]\frac{-q}{2}, \frac{q}{2}]$ by $\chi_{\alpha,q}$, where $\sigma = \alpha \cdot q$. That is,

$$\Pr[X = x] = \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} p'(t)dt \tag{2}$$

We note that, in our cases of interest, we can explicitly compute $\Pr[X = x]$ because both q and k are poly(n).

We state a straightforward lemma which will be useful in our computations later.

Lemma 1. *Let $X_0, \ldots, X_{m-1}$ be independent random variables, with $X_i \sim \mathcal{N}(\mu, \sigma^2)$. Then their sum $X = \sum_{i=0}^{m-1} X_i$ is also normally distributed, with $X \sim \mathcal{N}(m\mu, m\sigma^2)$.*

In the case of X_i following a discrete Gaussian distribution, it does not necessarily follow that a sum of such random variables is distributed in a way analogous to the statement above. However, throughout this work, we assume that this does hold i.e., that Lemma 1 applies to the discrete Gaussian case - while we do not know how to prove this, this assumption causes no apparent discrepancies in our experimental results. For a detailed discussion on sums of discrete Gaussian random variables, the interested reader is referred to [1].

COMPUTATIONAL MODEL. We express concrete costs as computational costs and storage requirements. We measure the former in $\mathbb{Z}_q$ operations and the latter in the number of $\mathbb{Z}_q$ elements requiring storage. However, as the hardness of LWE is related to the quantity $n \log q$ [10], relying on these measures would render results for different instances incommensurable. We hence normalise these magnitudes by considering "bit-operations" where one ring operation in $\mathbb{Z}_q$ is equivalent to $\log_2 q$ such bit operations. The specific multiplier $\log_2 q$ is derived from the fact that the majority of operations are additions and subtractions in $\mathbb{Z}_q$ as opposed to multiplications in $\mathbb{Z}_q$. In particular, we ignore the cost of "book keeping" and of fixed-precision floating point operations occuring during the algorithm (where the precision is typically a small multiple of n, cf. Section 4).

We make the assumption that we have unrestricted access to an LWE oracle, allowing us to obtain a large number of independent LWE samples which may not be available in practise. This assumption is usually made for combinatorial algorithms and the Arora-Ge algorithm, while lattice reduction algorithms usually require only a small number of LWE samples. However, as we discuss later, the optimal strategies for employing lattice based approaches for solving LWE appear to require executing a large number of small-advantage executions, each requiring independent LWE samples. While the cryptosystems considered in this work do not provide such an LWE oracle it is known [30] that given roughly $n \log q$ LWE samples one can produce many more LWE samples at the cost of an increase in the noise through inter-addition. While employing these approaches would render our proofs inapplicable, it is assumed that in practice similar results would

still hold. Similar notions (in the case of LPN) were considered in [14], although, as in this work, the authors did not analyse the impact of these steps.

NOTATION. We always start counting at zero, and denote vectors in bold. Given a vector $\mathbf{a}$, we denote by $\mathbf{a}_{(i)}$ the i-th entry in $\mathbf{a}$, i.e., a scalar, and by $\mathbf{a}_{(i,j)}$ the subvector of $\mathbf{a}$ spanning the entries at indices $i, \ldots, j-1$. When given a list of vectors, we index its elements by subscript, e.g., $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2$, to denote the first three vectors of the list. When we write $(\mathbf{a}_i, c_i)$ we always mean the output of an oracle which should be clear from the context. In particular, $(\mathbf{a}_i, c_i)$ does not necessarily refer to samples following the distribution $L_{\mathbf{s},\chi}$.

3 The BKW Algorithm

The BKW algorithm was proposed by Blum, Kalai and Wasserman [9] as a method for solving the LPN problem, with sub-exponential complexity, requiring $2^{\mathcal{O}(n/\log n)}$ samples and time. The algorithm can be adapted for tackling Search- and Decision-LWE, with complexity $2^{\mathcal{O}(n)}$, when the modulus is taken to be polynomial in n.

To describe and analyse the BKW algorithm we use terminology and intuitions from linear algebra. Recall that noise-free linear systems of equations are solved by (a) transforming them into a triangular shape, (b) recovering a candidate solution in one variable for the univariate linear equation produced and (c) extending this solution by back substitution. Similarly, if we are only interested in *deciding* whether a linear system of equations does have a common solution, the standard technique is to produce a triangular basis and express other rows as linear combinations of this basis, i.e., to attempt to reduce them to zero.

The BKW algorithm – when applied to Search-LWE – can be viewed as consisting of three stages somewhat analogous to those of linear system solving:

(a) sample reduction is a form of Gaussian elimination which, instead of treating each component independently, considers 'blocks' of b components per iteration, where b is a parameter of the algorithm.
(b) hypothesis testing tests candidate sub-solutions to recover components of the secret vector $\mathbf{s}$.
(c) back substitution such that the whole process can be continued on a smaller LWE instance.

On a high-level, to aid intuition, if the standard deviation of χ was zero and hence all equations were noise-free, we could obviously recover $\mathbf{s}$ by simple Gaussian elimination. When we have non-zero noise, however, the number of row-additions conducted during Gaussian elimination result in the noise being 'amplified' to such levels that recovery of $\mathbf{s}$ would generally be impossible. Thus the motivation behind the BKW algorithm can be thought of as using a greater number of rows but eliminating many variables with single additions of rows rather than just one. If we can perform a Gaussian elimination-like reduction of the sample matrix using few enough row additions, the resulting noise in the system is still 'low enough' to allow us to recover one or a few components of $\mathbf{s}$ at a time. While, mainly for convenience of analysis, we choose a nested-oracle approach below to define the algorithm, the above intuitive approach is essentially equivalent.

The way we study the complexity of the BKW algorithm for solving the LWE problem is closely related to the method described in [14]: given an oracle that returns samples according to the probability distribution $L_{\mathbf{s},\chi}$, we use the algorithm's first stage to construct an oracle returning samples according to another distribution, which we call $B_{\mathbf{s},\chi,a}$, where $a = \lceil n/b \rceil$ denotes the number of 'levels' of addition. The complexity of the algorithm is related to the number of operations performed in this transformation, to obtain the required number of samples for hypothesis testing.

We now study the complexity of the first stage of the BKW algorithm.

3.1 Sample Reduction

Given $n \in \mathbb{Z}$, select a positive integer $b \leq n$ (the window width), and let $a := \lceil n/b \rceil$ (the addition depth). Given an LWE oracle (which by abuse of notation, we will also denote by $L_{\mathbf{s},\chi}$), we denote by $B_{\mathbf{s},\chi,\ell}$ a related oracle which outputs samples where the first $b \cdot \ell$ coordinates of each $\mathbf{a}_i$ are zero, generated under the distribution (which again by abuse of notation, we denote by $B_{\mathbf{s},\chi,\ell}$) obtained as follows:

- if $\ell = 0$, then $B_{\mathbf{s},\chi,0}$ is simply $L_{\mathbf{s},\chi}$;
- if $0 < \ell < a$, the distribution $B_{\mathbf{s},\chi,\ell}$ is obtained by taking the difference of two vectors from $B_{\mathbf{s},\chi,\ell-1}$ that agree on the elements $(\mathbf{a}_{((\ell-1)\cdot b)}, \mathbf{a}_{((\ell-1)\cdot b+1)}, \ldots, \mathbf{a}_{(\ell \cdot b-1)})$.

We can then describe the first stage of the BKW algorithm as the (recursively constructed) series of sample oracles $B_{\mathbf{s},\chi,\ell}$, for $0 \leq \ell < a$. Indeed, we define $B_{\mathbf{s},\chi,0}$ as the oracle which simply returns samples from $L_{\mathbf{s},\chi}$, while $B_{\mathbf{s},\chi,\ell}$ is constructed from $B_{\mathbf{s},\chi,\ell-1}$, for $\ell \geq 1$. We will make use of a set of tables T (maintained across oracle calls) to store (randomly-chosen) vectors that will be used to reduce samples arising from our oracles. More explicitly, given a parameter $b \leq n$ for the window width, and letting $a = \lceil n/b \rceil$, we can describe the oracle $B_{\mathbf{s},\chi,\ell}$ as follows:

1. For $\ell = 0$, we can obtain samples from $B_{\mathbf{s},\chi,0}$ by simply calling the LWE oracle $L_{\mathbf{s},\chi}$ and returning the output.
2. For $\ell = 1$, we repeatedly query the oracle $B_{\mathbf{s},\chi,0}$ to obtain (at most) $(q^b - 1)/2$ samples $(\mathbf{a}, c)$ with distinct non-zero vectors for the first b coordinates of $\mathbf{a}$. We only collect $(q^b - 1)/2$ such vectors because we exploit the symmetry of $\mathbb{Z}_q$ and that of the noise distribution. We use these samples to populate the table T^1, indexed by the first b entries of $\mathbf{a}$. We store $(\mathbf{a}, c)$ in the table. During this course of this population, whenever we obtain a sample $(\mathbf{a}', c')$ from $B_{\mathbf{s},\chi,0}$, if either the first b entries of $\mathbf{a}'$ (resp. their negation) match the first b entries of a vector $\mathbf{a}$ such that the pair $(\mathbf{a}, c)$ is already in T^1, we return $(\mathbf{a}' \pm \mathbf{a}, c' \pm c)$, as a sample from $B_{\mathbf{s},\chi,1}$. Note that, if the first b entries in $\mathbf{a}'$ are zero, we return $(\mathbf{a}', c')$ as a sample from $B_{\mathbf{s},\chi,1}$. Further calls to the oracle $B_{\mathbf{s},\chi,1}$ proceed in a similar manner, but using (and potentially adding entries to) the same table T^1.
3. For $1 < \ell < a$, we proceed as above: we make use of the table T^ℓ (constructed by calling $B_{\mathbf{s},\chi,\ell-1}$ up to $(q^b - 1)/2$ times) to reduce any output sample from $B_{\mathbf{s},\chi,\ell-1}$ which has the b entries in its ℓ-th block already in T^ℓ, to generate a sample from $B_{\mathbf{s},\chi,\ell}$.

Pseudo-code for the oracle $B_{\mathbf{s},\chi,\ell}$, for $0 < \ell < a$, is given in Algorithm 1 (the case $\ell = a$ will be discussed below).

Input: b – an integer $0 < b \leq n$
Input: ℓ – an integer $0 < \ell < a$
begin
  $T^{\ell} \leftarrow$ array indexed by $\mathbb{Z}_q^b$ maintained across all runs of $B_{\mathbf{s},\chi,\ell}$;
  query $B_{\mathbf{s},\chi,\ell-1}$ to obtain $(\mathbf{a}, c)$;
  if $\mathbf{a}_{(b\cdot(\ell-1),b\cdot\ell)}$ *is all zero vector* **then**
    return $(\mathbf{a}, c)$;
  while $T^{\ell}_{\mathbf{a}_{(b\cdot(\ell-1),b\cdot\ell)}} = \varnothing$ **and** $T^{\ell}_{-\mathbf{a}_{(b\cdot(\ell-1),b\cdot\ell)}} = \varnothing$ **do**
    $T^{\ell}_{\mathbf{a}_{(b\cdot(\ell-1),b\cdot\ell)}} \leftarrow (\mathbf{a}, c)$;
    query $B_{\mathbf{s},\chi,\ell-1}$ to obtain $(\mathbf{a}, c)$;
    if $\mathbf{a}_{(b\cdot(\ell-1),b\cdot\ell)}$ *is all zero vector* **then**
      return $(\mathbf{a}, c)$;
  if $T^{\ell}_{\mathbf{a}_{(b\cdot(\ell-1),b\cdot\ell)}} \neq \varnothing$ **then**
    $(\mathbf{a}', c') \leftarrow T^{\ell}_{\mathbf{a}_{(b\cdot(\ell-1),b\cdot\ell)}}$;
    return $(\mathbf{a} - \mathbf{a}', c - c')$;
  else
    $(\mathbf{a}', c') \leftarrow T^{\ell}_{-\mathbf{a}_{(b\cdot(\ell-1),b\cdot\ell)}}$;
    return $(\mathbf{a} + \mathbf{a}', c + c')$;

Algorithm 1: $B_{\mathbf{s},\chi,\ell}$ for $0 < \ell < a$

Then, given an LWE oracle $L_{\mathbf{s},\chi}$ outputting samples of the form $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$, where $\mathbf{a}, \mathbf{s} \in \mathbb{Z}_q^n$, the oracle $B_{\mathbf{s},\chi,a-1}$ can be seen as another LWE oracle outputting samples of the form $(\mathbf{a}', \langle \mathbf{a}', \mathbf{s}' \rangle + e')$, where $\mathbf{a}', \mathbf{s}' \in \mathbb{Z}_q^k$, with $k = n \bmod b$, if b does not divide n, or $k = b$ otherwise, and e' is generated with a different distribution (related to the original error distribution and the value a). The vector $\mathbf{s}'$ is defined to be the last k components of $\mathbf{s}$. For the remainder of this section we will assume that $n \bmod b = 0$, and therefore $k = b$ (this is done to simplify the notation, but all the results obtained can be easily adapted when the last block has length $k < b$).

We note that, in our analysis below, we make the assumption for simplicity that all tables are completely filled during the elimination stage of the algorithm, thus giving conservative time and space bounds. In practise, especially in the final tables, this will not be the case and birthday-paradox arguments could be applied to derive a more realistic (lower) complexity. Typically, if the number of samples required for hypothesis-testing is small, the birthday paradox implies that the storage required for the final table can be reduced by a square-root factor.

Moreover, in this work we introduce an additional parameter $d \leq b$ which does not exist in the original BKW algorithm [9]. This parameter is used to reduce the number of hypotheses that need to be tested in the second stage of the algorithm, and arises from the fact we work with primes $q > 2$. At times, instead of working with the last block of length b, which could lead to potentially exponentially many hypotheses q^b to be tested, we may employ a final reduction phase – $B_{\mathbf{s},\chi,a}$ – to reduce the samples to $d < b$ non-zero entries in $\mathbf{a}$. Thus d will represent the number of components in our final block, i.e., the number of elements of the secret over which we conduct hypothesis tests. So after running the $B_{\mathbf{s},\chi,a-1}$ algorithm, we may decide to split the final block to obtain vectors over $\mathbb{Z}_q^d$. If so, we run the reduction function described above once more, which we will denote by $B_{\mathbf{s},\chi,a}$. In the simple case where we do not split the last block (i.e., $d = b$), we adopt the convention that we will also call the $B_{\mathbf{s},\chi,a}$ function, but it will perform no extra action (i.e., it simply calls $B_{\mathbf{s},\chi,a-1}$). Thus we have that for a choice of $0 \leq d \leq b$, the oracle $B_{\mathbf{s},\chi,a}$ will output samples of the form $(\mathbf{a}', \langle \mathbf{a}', \mathbf{s}' \rangle + e')$, where $\mathbf{a}', \mathbf{s}' \in \mathbb{Z}_q^d$. We pick $d = 0$ in the decision variant.

On choosing b and d. In general, as discussed above, choosing the parameter d to be small (e.g. 1 or 2) leads to the best results. However, in general one could also relax the condition $d \leq b$ to $d \leq n$ where $d = n$ is equivalent to straight-forward exhaustive search. Finally, a natural question is – should all the blocks be of equal length or should some be shorter than others? Intuitively, choosing blocks which are all of equal size (or as close as possible) appears to be the optimal strategy, though we do not formally investigate this here. To ease the presentation, we assume throughout this paper that this is the case.

From the constructions above, it follows that the cost of one call to $B_{\mathbf{s},\chi,\ell}$ is at most $\lceil q^b/2 \rceil$ calls to $B_{\mathbf{s},\chi,\ell-1}$. We also need at most one addition of two outputs of $B_{\mathbf{s},\chi,\ell-1}$, which have the first $b \cdot \ell$ entries in common. This operation requires $n+1-b\cdot\ell$ additions in $\mathbb{Z}_q$. Furthermore, since we maintain T^ℓ across different runs of $B_{\mathbf{s},\chi,\ell}$, this cost is amortised across different runs of $B_{\mathbf{s},\chi,\ell}$. Hence, when $\ell > 0$ the cost of calling $B_{\mathbf{s},\chi,\ell}$ a total of m times is upper bounded by:

$$m + \frac{q^b - 1}{2} \text{ calls to } B_{\mathbf{s},\chi,\ell-1} \text{ and } m \text{ additions of outputs of } B_{\mathbf{s},\chi,\ell-1}.$$

Overall we obtain Lemma 2.

Lemma 2. *Let $n \geq 1$ be the dimension of the LWE secret vector, q be a positive integer, and $d, b \in \mathbb{Z}$ with $1 \leq d \leq b \leq n$, and define $a = \lceil n/b \rceil$. The cost of calling $B_{\mathbf{s},\chi,a}$, which returns samples $(\mathbf{a}_i, c_i)$ with at most the d rightmost entries of $\mathbf{a}_i$ non-zero, m times is upper bounded by*

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

additions in $\mathbb{Z}_q$ and $a \cdot \left\lceil \frac{q^b}{2} \right\rceil + m$ calls to $L_{\mathbf{s},\chi}$.

Proof. We may think of the $B_{\mathbf{s},\chi,\ell}$ oracles as constructing the matrix

$$B = \begin{pmatrix}
T^1 & & & & & & & & & \\
0 & \dots & 0 & T^2 & & & & & & \\
0 & \dots & 0 & 0 & \dots & 0 & T^3 & & & \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & & \\
0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots 0 & T^{a-1} & \\
0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots 0 & 0 & T^a \\
0 & \dots & 0 & 0 & \dots & 0 & 0 & \dots 0 & \dots\ 0 & M
\end{pmatrix}$$

where, for $1 \leq i \leq a-1$, T^i represents a submatrix of $(q^b-1)/2$ rows and $n+1-b\cdot(i-1)$ columns (the $+1$ accounts for the c_i column). According to our convention, T^a is either a submatrix of $(q^{b-d}-1)/2$ rows and $n+1-b\cdot(a-1)$ columns if we split the last block (i.e. $d < b$), or an empty matrix. Finally M represents a submatrix with m rows and $d+1$ columns. The matrix B has therefore at most $(a-1)\cdot((q^b-1)/2) + ((q^{b-d}-1)/2) + m < a \cdot \left\lceil \frac{q^b}{2} \right\rceil + m$ rows and hence needs as many calls to $L_{\mathbf{s},\chi}$ to be constructed. This proves the second claim.

For the upper-bound on the number of additions necessary in $\mathbb{Z}_q$, we have the following (we treat the worst case, where full construction of all T tables is necessary before obtaining any samples from $B_{\mathbf{s},\chi,a}$): The construction of a T-tables is required, only $a-1$ (at most) of which require additions.

1. The construction of table T^1 requires 0 ring additions.

2. The construction of table T^2 requires at most $((q^b-1)/2)\cdot(n+1-b)$ additions.
3. The construction of table T^3 requires at most $((q^b-1)/2)\cdot((n+1-b)+(n+1-2b))$ additions.
4. In general, for $2<i<a$, the construction of table T^i requires at most

$$\left(\frac{q^b-1}{2}\right)\cdot\left((i-1)\cdot(n+1)-\sum_{j=1}^{i-1}j\cdot b\right)=\left(\frac{q^b-1}{2}\right)\cdot(i-1)\cdot\left((n+1)-\frac{i}{2}\cdot b\right).$$

5. The construction of T^a - the above expression is an upper bound for $i=a$.
6. Thus, the construction of all the T^i tables requires at most

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

   additions in $\mathbb{Z}_q$.
7. Now, for the construction of our m final samples, the construction of each of these samples requires at most

$$\begin{aligned}(n+1-b)+(n+1-2b)+\ldots+(n+1-a\cdot b)&=\sum_{i=1}^{a}(n+1-ib)\\&<a\cdot\left((n+1)-\frac{n}{2}\right)\\&=\frac{a}{2}\cdot(n+2)\end{aligned}$$

   additions (in $\mathbb{Z}_q$).
8. Thus, the number of additions (in $\mathbb{Z}_q$) incurred through calling $B_{\mathbf{s},\chi,a}$ m times is upper-bounded by:

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

and this concludes the proof of the lemma. □

The memory requirements for storing the tables T^i are established in Lemma 3 below.

Lemma 3. *Let $n\geq 1$ be the dimension of the secret, q be a positive integer, and $d,b\in\mathbb{Z}$ with $1\leq d\leq b\leq n$, and define $a=\lceil n/b\rceil$. The memory required to store the table T^i is upper bounded by*

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

$\mathbb{Z}_q$ elements, each of which requires $\lceil\log_2(q)\rceil$ bits of storage.

Proof. The table T^1 has $\frac{q^b}{2}$ entries each of which holds $n+1$ elements of $\mathbb{Z}_q$. The table T^2 has the same number of entries but holds on $n+1-b$ elements of $\mathbb{Z}_q$. Overall, we get that all tables together hold

$$\begin{aligned}\sum_{i=1}^{a}\left(\frac{q^b}{2}\right)\cdot\left(n+1-(i-1)b\right)&=\left(\frac{q^b}{2}\right)\sum_{i=1}^{a}n+1-(i-1)b\\&=\left(\frac{q^b}{2}\right)\cdot a\cdot\left(n+1-b\frac{a-1}{2}\right)\end{aligned}$$

$\mathbb{Z}_q$ elements. □

Note however that, while the original LWE oracle $L_{\mathbf{s},\chi}$ may output zero vectors (which offer no information for the hypothesis tests in the search variant) with probability q^{-n}, the oracle $B_{\mathbf{s},\chi,a}$ may output such zero vectors with noticeable probability. In particular, calling $B_{\mathbf{s},\chi,a}$ m times does not guarantee that we get m samples with non-zero coefficients in $\mathbf{a}_i$. The probability of obtaining a zero vector from $B_{\mathbf{s},\chi,a}$ is $\frac{1}{q^d}$, and thus expect to have to call the oracle $B_{\mathbf{s},\chi,a}$ around $q^d/(q^d-1)\cdot m$ times to obtain $\approx m$ useful samples with good probability.

3.2 Hypothesis Testing

To give concrete estimates for the time and data complexity of solving a Search-LWE instance using BKW, we formulate the problem of solving an LWE instance as the problem of distinguishing between two different distributions. Assume we have m samples in $\mathbb{Z}_q^d \times \mathbb{Z}_q$ from $B_{\mathbf{s},\chi,a}$. It follows that we have $\mathbb{Z}_q^d$ hypotheses to test. In what follows, we examine each hypothesis in turn and derive a hypothesised set of noise values as a result (each one corresponding to one of the m samples). We show that if we have guessed incorrectly for the subvector $\mathbf{s}'$ of $\mathbf{s}$ then the distribution of these hypothesised noise elements will be (almost) uniform while if we guess correctly then these hypothesised noise elements will be distributed according to χ_a. That is, if we have that the noise distribution associated with samples from $L_{\mathbf{s},\chi}$ is $\chi = \chi_{\alpha,q}$, then it follows from Lemmas 1 and 2 that the noise distribution of samples obtained from $B_{\mathbf{s},\chi,\ell}$ follows $\chi_{\sqrt{2^\ell}\alpha,q}$ if all inputs are independent, i.e., we are adding 2^ℓ discrete Gaussians and produce a discrete Gaussian with standard deviation increased by a factor of $\sqrt{2^\ell}$. For the sake of simplicity, we denote this distribution by χ_ℓ in the remainder of this work and also assume that the oracle $B_{\mathbf{s},\chi,a}$ performs non-trivial operations on the output of $B_{\mathbf{s},\chi,a-1}$, i.e., the oracle $B_{\mathbf{s},\chi,a}$ performs a further reduction step. In other words we assume that the final oracle $B_{\mathbf{s},\chi,a}$ results in a further increase in the standard deviation of the noise distribution associated with the final samples which are used to test hypotheses for elements of $\mathbf{s}$. We hence make the following assumption in this section:

Assumption 1 *If we let* $\mathbf{s}' := \mathbf{s}_{(n-d,n)} = (\mathbf{s}_{(n-d)}, \ldots, \mathbf{s}_{(n-1)})$*, then the output of* $B_{\mathbf{s},\chi,a}$ *is generated as*

$$\mathbf{a} \leftarrow_\$ \mathbb{Z}_q^d, e \leftarrow_\$ \chi_a : (\mathbf{a}, \langle \mathbf{a}, \mathbf{s}' \rangle + e).$$

Remark 1. This section only refers to the Search-LWE problem in which we assume q is prime for ease of analysis and exposition. This restriction does not apply to our results below on the decision variant.

For our hypothesis-testing strategies, we think of each of the samples returned by $B_{\mathbf{s},\chi,a}$ as giving rise to many equations

$$f_i = -c_i \pm j + \sum_{k=0}^{d-1} (\mathbf{a}_i)_{(k)} x_{(k)} \text{ for } 0 \leq j < q/2.$$

Given a number of these samples, in order to get an estimate for $\mathbf{s}'$, we run through q^d hypotheses and compute an array of scores S indexed by the possible guesses in $\mathbb{Z}_q^d$. That is, a function W assigns a weight to elements in $\mathbb{Z}_q$ which represent the noise under the hypothesis $\mathbf{s}' = \mathbf{v}$. For each guess $\mathbf{v}$ we sum over the weighted noises $W(-c_i + \sum_{k=0}^{d-1} (\mathbf{a}_i)_{(k)} \cdot v_{(k)})$. If W is such that the counter $S_\mathbf{v}$ grows proportionally to the likelihood that $\mathbf{v}$ is the correct guess, then the counter $S_{\mathbf{s}'}$ will grow fastest. Pseudo-code is given in Algorithm 2.

Input: F – a set of m samples following $B_{\mathbf{s},\chi,a}$
Input: W – a weight function mapping members of $\mathbb{Z}_q$ to real numbers
begin
  $S \leftarrow$ array filled with zeros indexed by $\mathbb{Z}_q^d$;
  for $\mathbf{v} \in \mathbb{Z}_q^d$ **do**
    $w_{\mathbf{v}} \leftarrow \emptyset$;
    for $f_i \in F$ **do**
      write f_i as $-c_i + \sum_{k=0}^{d-1}(\mathbf{a}_i)_{(k)} \cdot x_{(k)}$;
      $j \leftarrow \langle \mathbf{a}_i, \mathbf{v} \rangle - c_i$;
      $w_{\mathbf{v}} \leftarrow w_{\mathbf{v}} \cup \{W(j)\}$;
    $S_{\mathbf{v}} \leftarrow \sum_{w_i \in w_{\mathbf{v}}} w_i/m$;
  return S

Algorithm 2: Analysing candidates.

Lemma 4. *Running hypothesis testing costs $m \cdot q^d$ operations in $\mathbb{Z}_q$.*

Proof. Evaluating $\langle \mathbf{a}_i, \mathbf{v} \rangle - c_i$ at some point in $\mathbb{Z}_q^d$ naively costs $2d$ operations in $\mathbb{Z}_q$ which implies an overall cost of $2d \cdot m \cdot q^d$. However, we can reorder the elements in $\mathbb{Z}_q^d$ such that the element at index h differs from the element at index $h+1$ by an addition of a unit vector in $\mathbb{Z}_q^d$. Evaluating $\langle \mathbf{a}_i, \mathbf{v} \rangle - c_i$ on all $\mathbb{Z}_q^d$ points ordered in such a way reduces to one operation in $\mathbb{Z}_q$: addition of $\mathbf{a}_{i,(j)}$ where j is the index at which two consecutive elements differ. Hence, Algorithm 2 costs $m \cdot q^d$ operations in $\mathbb{Z}_q$. □

Recall that χ_a is the distribution of the errors under a right guess. Now, let $\mathcal{U}_a$ denote the distribution of errors under a wrong guess $\mathbf{v} \neq \mathbf{s}'$. By the Neyman-Pearson Lemma, the most powerful test of whether samples follow one of two known distributions is the log-likelihood ratio.

Hence, for j, with $\lceil -q/2 \rceil \leq j \leq \lfloor q/2 \rfloor$, we set:

$$W(j) := \log_2 \left(\frac{\Pr[e \leftarrow_\$ \chi_a : e = j]}{\Pr[e \leftarrow_\$ \mathcal{U}_a : e = j]} \right). \tag{3}$$

Next, we establish the relation between $\tilde{p}_j := \Pr[e \leftarrow_\$ \mathcal{U}_a : e = j]$ and $p_j := \Pr[e \leftarrow_\$ \chi_a : e = j]$.

Lemma 5. *Given a wrong guess $\mathbf{v}$ for $\mathbf{s}'$, for each element $f_i = -c_i + \sum_{k=0}^{d-1}(\mathbf{a}_i)_{(k)} x_{(k)} \in \mathbb{Z}_q[x]$, with $c_i = \langle \mathbf{a}_i, \mathbf{s}' \rangle - e_i$, the probability of error j appearing is*

$$\tilde{p}_j := \Pr[e \leftarrow_\$ \mathcal{U}_a : e = j] = \frac{q^{d-1} - p_j}{q^d - 1} \tag{4}$$

if q is prime.

Proof. We write $\mathbf{v} = \mathbf{s}' + \mathbf{t}$. Since $\mathbf{v}$ is a wrong guess, we must have $\mathbf{t} \neq \mathbf{0}$. For a fixed $\mathbf{s}'$ and for $\mathbf{a}_i, \mathbf{t} \neq \mathbf{0}$, it holds that:

$$\begin{aligned} \tilde{p}_j &= \Pr[e \leftarrow_\$ \mathcal{U}_a : e = j] = \Pr[\langle \mathbf{a}_i, \mathbf{v} \rangle - c_i = j] = \Pr[\langle \mathbf{a}_i, \mathbf{t} \rangle - e_i = j] \\ &= \sum_{y=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} \left(\Pr[\sum_{k=0}^{d-1} \mathbf{a}_{i(k)} \mathbf{t}_{(k)} = y] \cdot \Pr[j + e_i = y] \right). \end{aligned}$$

This is equal to:

$$\sum_{y=\lceil -q/2 \rceil}^{y=-1} \left(\Pr[\sum_{k=0}^{d-1} \mathbf{a}_{i(k)} \mathbf{t}_{(k)} = y] \cdot \Pr[j + e_i = y] \right)$$
$$+ \sum_{y=1}^{y=\lfloor q/2 \rfloor} \left(\Pr[\sum_{k=0}^{d-1} \mathbf{a}_{i(k)} \mathbf{t}_{(k)} = y] \cdot \Pr[j + e_i = y] \right)$$
$$+ \Pr[\sum_{k=0}^{d-1} \mathbf{a}_{i(k)} \mathbf{t}_{(k)} = 0] \cdot \Pr[e_i = -j].$$

Now, since our q is prime, for any two non-zero elements $y, z \in \mathbb{Z}_q$, we have that:

$$\Pr[\sum_{k=0}^{d-1} \mathbf{a}_{i(k)} \mathbf{t}_{(k)} = y] = \Pr[\sum_{k=0}^{d-1} \mathbf{a}_{i(k)} \mathbf{t}_{(k)} = z].$$

We denote this probability by $p_{(\neq 0)}$. Conversely, we denote the probability of obtaining $\langle \mathbf{a}_i, \mathbf{t} \rangle = 0$ by $p_{(=0)}$. Then we clearly have $p_{(\neq 0)} = \frac{1 - p_{(=0)}}{q-1}$. Thus we can write:

$$\tilde{p}_j = \left(p_{(\neq 0)} \sum_{y=\lceil -q/2 \rceil}^{y=-1} \Pr[j + e_i = y] \right) + \left(p_{(\neq 0)} \sum_{y=\lceil -q/2 \rceil}^{y=-1} \Pr[j + e_i = y] \right)$$
$$+ \Pr[\sum_{k=0}^{d-1} \mathbf{a}_{i(k)} \mathbf{t}_{(k)} = 0] \cdot \Pr[e_i = -j].$$

Note that

$$1 - p_{-j} = \Pr[e_i \leftarrow_{\$} \chi_a : e_i \neq -j] = \sum_{y=\lceil -q/2 \rceil}^{y=-1} \Pr[j + e_i = y] + \sum_{y=1}^{\lfloor q/2 \rfloor} \Pr[j + e_i = y].$$

By definition, $p_{-j} = p_j$. Thus:

$$\tilde{p}_j = p_{(\neq 0)} \cdot (1 - p_j) + p_{(=0)} \cdot p_j.$$

Now, to determine $p_{(=0)}$, the exclusion of zero-vectors from the set of all possible dot products reduces the number of zero dot products from $q^d + q^{2d-1} - q^{d-1}$ to $q^{2d-1} - q^{d-1} - q^d + 1$. Thus we have that the probability of obtaining a zero dot product is:

$$p_{(=0)} = \frac{q^{2d-1} - q^{d-1} - q^d + 1}{q^{2d} - 2q^d + 1} = \frac{q^{d-1} - 1}{q^d - 1}.$$

Thus, we have:

$$\tilde{p}_j = (1 - p_j) \cdot \frac{q^{d-1}}{q^d - 1} + p_j \cdot \frac{q^{d-1} - 1}{q^d - 1} = \frac{q^{d-1} - p_j}{q^d - 1}$$

as required. □

For the final backsubstitution stage, we wish to ensure that the score for the correct guess $\mathbf{v} = \mathbf{s}'$ is highest among the entries of S. Thus, what remains to be established is the size $m = |F|$ needed such that the score for the right guess $\mathbf{v} = \mathbf{s}'$ is the highest. Under our sample independence assumptions, by the Central Limit theorem, the distribution of $S_{\mathbf{v}}$ approaches a Normal distribution as m increases. Hence, for sufficiently large m we may approximate the discrete distribution $S_{\mathbf{v}}$ by a normal distribution [8]. If $\mathcal{N}(\mu, \sigma^2)$ denotes a Normal distribution with mean μ and standard deviation σ we denote the distribution for $\mathbf{v} = \mathbf{s}'$ by $D_c = \mathcal{N}(\mathrm{E}_c, \mathrm{Var}_c)$ and for $\mathbf{v} \neq \mathbf{s}'$ by $D_w = \mathcal{N}(\mathrm{E}_w, \mathrm{Var}_w)$.

Establishing m hence first of all means establishing $\mathrm{E}_c, \mathrm{E}_w, \mathrm{Var}_c$ and Var_w. We start with E_c.

Lemma 6. *Let $(\mathbf{a}_0, c_0), \ldots, (\mathbf{a}_{m-1}, c_{m-1})$ be samples following $B_{\mathbf{s},\chi,a}$, q be a positive integer, $\mathbf{v} \in \mathbb{Z}_q^d$, $p_j := \Pr(e \leftarrow_\$ \chi_a : e = j)$, $w_j := W(j)$ and $S_\mathbf{v} = \frac{1}{m}\sum_{i=0}^{m-1} W(\langle \mathbf{a}_i, \mathbf{v}\rangle - c_i)$. When $\mathbf{v} = \mathbf{s}'$, $\mathrm{E}(S_\mathbf{v})$ is given by:*

$$\mathrm{E}_c = \mathrm{E}(S_\mathbf{v} \mid \mathbf{v} = \mathbf{s}') = \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} p_j w_j = p_0 w_0 + 2 \cdot \sum_{j=1}^{j=\lfloor q/2 \rfloor} p_j w_j. \tag{5}$$

Proof. First, we remark that:

$$\Pr[\langle \mathbf{a}_i, \mathbf{s}'\rangle = c_i + u] = \Pr[\langle \mathbf{a}_i, \mathbf{s}'\rangle = \langle \mathbf{a}_i, \mathbf{s}'\rangle + e_i + u] = \Pr[-e_i = u] = p_u.$$

The expected value for $S_\mathbf{v}$ in the case of a correct guess is then given by:

$$\mathrm{E}_c := \mathrm{E}(S_\mathbf{v} \mid \mathbf{v} = \mathbf{s}') = \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} \Pr[e_i = j] \cdot W(j) = \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} p_j w_j.$$

Finally, for all $j, 1 \leq j \leq \lfloor q/2 \rfloor$, we have $p_{-j} w_{-j} = p_j w_j$. Thus:

$$\sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} p_j w_j = p_0 w_0 + 2 \cdot \sum_{j=1}^{j=\lfloor q/2 \rfloor} p_j w_j.$$

□

We now examine $\mathrm{E}_w = \mathrm{E}(S_\mathbf{v} \mid \mathbf{v} \neq \mathbf{s}')$. To begin with, we fix a wrong guess $\mathbf{v}$, such that $\mathbf{v} = \mathbf{s}' + \mathbf{t}$ with $\mathbf{t} \neq 0$.

Lemma 7. *Let $(\mathbf{a}_0, c_0), \ldots, (\mathbf{a}_{m-1}, c_{m-1})$ be samples following $B_{\mathbf{s},\chi,a}$, q be a positive integer, $\mathbf{v} \in \mathbb{Z}_q^d$, $\tilde{p}_j := \Pr(e \leftarrow_\$ \mathcal{U}_a : e = j)$, $p_j := \Pr(e \leftarrow_\$ \chi_a : e = j)$, $w_j := W(j)$, and $S_\mathbf{v} = \frac{1}{m}\sum_{i=0}^{m-1} W(\langle \mathbf{a}_i, \mathbf{v}\rangle - c_i)$. If $\mathbf{v} \neq \mathbf{s}'$, we have:*

$$\mathrm{E}_w = \mathrm{E}(S_\mathbf{v} \mid \mathbf{v} \neq \mathbf{s}') = \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} \tilde{p}_j w_j = \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} \frac{q^{d-1} - p_j}{q^d - 1} w_j. \tag{6}$$

Since the proof of Lemma 7 is analogous to Lemma 6 we omit it here. We now look at the variances Var_c and Var_w.

Lemma 8. *Let $(\mathbf{a}_0, c_0), \ldots, (\mathbf{a}_{m-1}, c_{m-1})$ be samples following $B_{\mathbf{s},\chi,a}$, q be a positive integer, $\mathbf{v} \in \mathbb{Z}_q^d$, $p_j := \Pr(e \leftarrow_\$ \chi_a : e = j)$, $\tilde{p}_j := \Pr(e \leftarrow_\$ \mathcal{U}_a : e = j)$, $w_j := W(j)$ and $S_\mathbf{v} = \sum_{i=0}^{m-1} \frac{1}{m} W(\langle \mathbf{a}_i, \mathbf{v}\rangle - c_i)$.*

If $\mathbf{v} = \mathbf{s}'$, then

$$\mathrm{Var}_c := \mathrm{Var}(S_\mathbf{v} \mid \mathbf{v} = \mathbf{s}') = \frac{1}{m} \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} p_j \cdot (w_j - \mathrm{E}_c)^2. \tag{7}$$

If $\mathbf{v} \neq \mathbf{s}'$, then

$$\mathrm{Var}_w := \mathrm{Var}(S_\mathbf{v} \mid \mathbf{v} \neq \mathbf{s}') = \frac{1}{m} \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} \tilde{p}_j \cdot (w_j - \mathrm{E}_w)^2. \tag{8}$$

Proof. In the case of $\mathbf{v} = \mathbf{s}'$ we have that for $m = 1$,

$$\mathrm{Var}_c = \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} p_j \cdot \big(w_j - \mathrm{E}_c\big)^2.$$

In the case of adding then normalising m samples we can use the fact that when adding random variables of zero covariance, the sum of the variances is the variance of the sum. Thus the variance in the case of adding m samples and normalising is given by:

$$\mathrm{Var}_c = m \cdot \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} p_j \cdot \left(\frac{w_j}{m} - \frac{\mathrm{E}_c}{m}\right)^2 = \frac{1}{m} \sum_{j=\lceil -q/2 \rceil}^{\lfloor q/2 \rfloor} p_j \cdot (w_j - \mathrm{E}_c)^2$$

A similar argument holds in the case of Var_w. □

Finally, given E_c, E_w, Var_c, and Var_w, we can estimate the rank of the right secret in dependence of the number of samples m considered. We denote by Y_h the random variable determined by the rank of a correct score $S_{\mathbf{s}'}$ in a list of h elements. Now, for a list of length q^d and a given rank $0 \leq r < q^d$, the probability of Y_{q^d} taking rank r is given by a binomial-normal compound distribution. Finally, we get Lemma 9, which essentially states that for whatever score the right secret gets, in order for it to have rank zero the remaining $q^d - 1$ secrets must have smaller scores.

Lemma 9. *Let E_c, E_w, Var_c and Var_w be as in Lemmas 6, 7 and 8. Let also Y_{q^d} be the random variable determined by the rank of a correct score $S_{\mathbf{s}'}$ in the list S of q^d elements. Then, the number of samples m required for Y_{q^d} to take rank zero with probability ϵ' is recovered by solving*

$$\epsilon' = \int_x \left[\frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x - \mathrm{E}_w}{\sqrt{2\mathrm{Var}_w}}\right)\right)\right]^{(q^d-1)} \cdot \left(\frac{1}{\sqrt{2\pi\mathrm{Var}_c}} e^{-\frac{(x-\mathrm{E}_c)^2}{2\mathrm{Var}_c}}\right) \mathrm{d}x,$$

for m.

Proof. Y_{q^d} follows a binomial-normal compound distribution given by $\Pr[Y_{q^d} = r] =$

$$\int_x \left(\binom{q^d - 1}{r} \cdot \Pr[e \leftarrow_\$ D_w : e \geq x]^r \cdot \Pr[e \leftarrow_\$ D_w : e < x]^{(q^d - r - 1)} \cdot \Pr[e \leftarrow_\$ D_c : e = x]\right) \mathrm{d}x.$$

Plugging in $r = 0$ and $\Pr[Y_{q^d} = r] = \epsilon'$ we get:

$$\begin{aligned}
\epsilon' &= \int_x \Pr[e \leftarrow_\$ D_w : e < x]^{(q^d-1)} \cdot \Pr[e \leftarrow_\$ D_c : e = x] \, \mathrm{d}x \\
&= \int_x \left[\frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x - \mathrm{E}_w}{\sqrt{2\mathrm{Var}_w}}\right)\right)\right]^{(q^d-1)} \cdot \left(\frac{1}{\sqrt{2\pi\mathrm{Var}_c}} e^{-\frac{(x-\mathrm{E}_c)^2}{2\mathrm{Var}_c}}\right) \mathrm{d}x
\end{aligned}$$

as required. □

Using Lemma 9 we can hence estimate the number of non-zero samples m we need to recover subvector $\mathbf{s}'$.

Remark 2. We note that Algorithm 2 not only returns an ordering of the hypotheses but also a score for each hypothesis. Hence, we can simply sample from $B_{\mathbf{s},\chi,a}$ until the distance between the first and second highest rated hypothesis is above a certain threshold.

3.3 Back Substitution

Given a candidate solution for $\mathbf{s}'$ which is correct with very high probability we can perform backsubstitution in our tables T^i similarly to solving a triangular linear system. It is easy to see that backsubstitution costs $2d$ operations per row. Furthermore, by Lemma 2 we have $a \cdot (\lceil q^b/2 \rceil)$ rows in all tables T^i.

After backsubstitution, we start the BKW algorithm again in stage one where all the tables T^i are already filled. To recover the next d components of $\mathbf{s}$ then, we ask for m fresh samples which are reduced using our modified tables T^i and perform hypothesis testing on these m samples.

3.4 Complexity of BKW

We can now state our main theorem.

Theorem 2 (Search-LWE). *Let $(\mathbf{a}_i, c_i)$ be samples following $L_{\mathbf{s},\chi}$, $0 < b \leq n$, $d \leq b$ parameters, $0 < \epsilon < 1$ the targeted success rate and q prime. Let $a = \lceil n/b \rceil$ and m be as in Lemma 9 when $\epsilon' = (\epsilon)^{1/\lceil n/d \rceil}$. Then, the expected cost of the BKW algorithm to recover $\mathbf{s}$ with success probability ϵ is*

$$\left(\frac{q^b - 1}{2}\right) \cdot \left(\frac{a(a-1)}{2} \cdot (n+1) - \frac{ba(a-1)}{4} - \frac{b}{6}\left((a-1)^3 + \frac{3}{2}(a-1)^2 + \frac{1}{2}(a-1)\right)\right) \tag{9}$$

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

$$\frac{q^d}{q^d - 1} \cdot \frac{\lceil \frac{n}{d} \rceil + 1}{2} \cdot m \cdot \left(\frac{a}{2} \cdot (n+2)\right) \tag{10}$$

additions/subtractions in $\mathbb{Z}_q$ to produce samples for hypothesis testing. For the hypothesis-testing step

$$\left\lceil \frac{n}{d} \right\rceil \cdot \left(m \cdot q^d\right) \tag{11}$$

arithmetic operations in $\mathbb{Z}_q$ are required and

$$\left(\left\lceil \frac{n}{d} \right\rceil + 1\right) \cdot d \cdot a \cdot \left\lceil \frac{q^b}{2} \right\rceil \tag{12}$$

operations in $\mathbb{Z}_q$ for backsubstitution. Furthermore,

$$a \cdot \left\lceil \frac{q^b}{2} \right\rceil + \frac{q^d}{q^d - 1} \cdot \left\lceil \frac{n}{d} \right\rceil \cdot m \tag{13}$$

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

$$\left(\frac{q^b}{2}\right) \cdot a \cdot \left(n + 1 - b\frac{a-1}{2}\right) \tag{14}$$

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

Proof. In order to recover $\mathbf{s}$ we need every run of stage 1 to be successful, hence we have $\epsilon = (\epsilon')^{\lceil n/d \rceil}$ and consequently $\epsilon' = (\epsilon)^{1/\lceil n/d \rceil}$.

Furthermore, we have:

– The cost of constructing the tables T^i in Equation (9) follows from Lemma 2.
– Lemma 2 and the fact that with probability $\frac{1}{q^d}$ the oracle $B_{\mathbf{s},\chi,a}$ returns an all-zero sample establish that to produce m non-zero samples for hypothesis testing, $\frac{q^d}{q^d-1} \cdot m \cdot \left(\frac{a}{2} \cdot (n+2)\right)$ operations are necessary. We need to produce m such samples $\left\lceil \frac{n}{d} \right\rceil$ times. However, as we proceed the number of required operations linearly approaches zero. Hence, we need $\frac{\left\lceil \frac{n}{d} \right\rceil + 1}{2} \cdot \frac{q^d}{q^d-1} \cdot m \cdot \left(\frac{a}{2} \cdot (n+2)\right)$ operations as in Equation (10).
– The cost of Algorithm 2 in Equation (11) which also is run $\left\lceil \frac{n}{d} \right\rceil$ times follows from Lemma 4.
– There are $a \cdot \left\lceil \frac{q^b}{2} \right\rceil$ rows in all tables T^i each of which requires $2d$ operations in backsubstitution. We need to run backsubstitution $\left\lceil \frac{n}{d} \right\rceil$ times, but each time the cost decreases linearly. From this follows Equation (12).
– The number of samples needed in Equation (13) follows from Lemma 2 and that with probability $\frac{1}{q^d-1}$ the oracle $B_{\mathbf{s},\chi,a}$ returns a sample which is useless to us. □
– The storage requirement in Equation (14) follows from Lemma 3.

We would like to express the complexity of the BKW algorithm as a function of n, q, α explicitly. In that regard, Theorem 2 does not deliver yet. However, from the fact that we can distinguish χ_a and $\mathcal{U}_a$ in subexponential time if the standard deviation $\sqrt{2^a}\alpha q < q/2$ (i.e, the standard deviation of the discrete Gaussian distribution over $\mathbb{Z}$ corresponding to χ_a), we can derive the following simple corollary eliminating m.

Corollary 2. *Let $(\mathbf{a}_i, c_i)$ be samples following $L_{\mathbf{s},\chi}$, set $a = \lfloor \log_2(1/(2\alpha)^2) \rceil$, $b = n/a$ and q a prime. Let d be a small constant $0 < d < \log_2(n)$. Assume α is such that $q^b = q^{n/a} = q^{n/\lfloor \log_2(1/(2\alpha)^2) \rceil}$ is superpolynomial in n. Then, given these parameters, the cost of the BKW algorithm to solve* Search-LWE *is*

$$\left(\frac{q^b-1}{2}\right) \cdot \left(\frac{a(a-1)}{2} \cdot (n+1)\right) + \left\lceil \frac{q^b}{2} \right\rceil \cdot \left(\left\lceil \frac{n}{d} \right\rceil + 1\right) \cdot d \cdot a + \mathrm{poly}(n) \approx \left(a^2 n\right) \cdot \frac{q^b}{2}$$

operations in $\mathbb{Z}_q$. Furthermore,

$$a \cdot \left\lceil \frac{q^b}{2} \right\rceil + \mathrm{poly}(n) \text{ calls to } L_{\mathbf{s},\chi} \text{ and storage of } \left(a \cdot \left\lceil \frac{q^b}{2} \right\rceil \cdot n\right) \text{ elements in } \mathbb{Z}_q \text{ are needed.}$$

Proof. From the condition $\sqrt{2^a} \cdot \alpha \cdot q < q/2$ follows that we must set $a = \log_2(1/(2\alpha)^2)$. If a is set this way we have that we can distinguish χ_a from $\mathcal{U}(\mathbb{Z}_q)$ in $\mathrm{poly}(n)$. Now, since q^b is superpoynomial in n we have that $m \leq q^b$ and Theorem 2 is dominated by terms involving q^b. □

In many cryptographic applications solving the Decision-LWE problem is equivalent to breaking the cryptographic assumption. Furthermore, in many such constructions q may not be a prime. Hence, we also establish the cost of distinguishing $L_{\mathbf{s},\chi}$ from random with a given success probability for arbitrary moduli q.

Corollary 3 (Decision-LWE). *Let $(\mathbf{a}_i, c_i)$ be samples following $L_{\mathbf{s},\chi}$, $0 < b \leq n$ be a parameter, $0 < \epsilon < 1$ the targeted success rate and $a = \lceil n/b \rceil$ the addition depth. Then, the expected cost of the BKW algorithm to distinguish $L_{\mathbf{s},\chi}$ from random with success probability ϵ is*

$$\left(\frac{q^b-1}{2}\right) \cdot \left(\frac{a(a-1)}{2} \cdot (n+1) - \frac{ba(a-1)}{4} - \frac{b}{6}\left((a-1)^3 + \frac{3}{2}(a-1)^2 + \frac{1}{2}(a-1)\right)\right) \tag{15}$$

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

$$m \cdot \left(\frac{a}{2} \cdot (n+2)\right) \text{ with } m = \epsilon / \exp\left(-\frac{\pi^2 \sigma^2 2^{a+1}}{q^2}\right) \tag{16}$$

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

$$a \cdot \left\lceil \frac{q^b}{2} \right\rceil + m \tag{17}$$

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

$$\left(\frac{q^b}{2}\right) \cdot a \cdot \left(n + 1 - b\frac{a-1}{2}\right) \tag{18}$$

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

Proof. No hypothesis testing, backsubstitution and accounting for all zero samples is necessary and hence any terms referring to those can be dropped from Theorem 2.

Choosing $m = \exp\left(-\frac{\pi^2\sigma^2 2^{a+1}}{q^2}\right)/\epsilon$ leads to a distinguishing advantage of ϵ (cf. [21]). □

4 Applications

In this section we apply Theorem 2 to various sets of parameters suggested in the literature. In order to compute concrete costs we rely on numerical approximations in various places such as the computation of p_j. We used $2n - 4n$ bits of precision for all computations, increasing this precision further did not appear to change our results. The solving step for m of Lemma 9 is accomplished by a simple search implemented in Sage [32]. As a subroutine of this search we rely on numerical integration which we performed using the mpmath library [20] as shipped with Sage. The Sage script used to derive all values in this section is available at [4].

In all cases below we always set $\epsilon = 0.99$ and $a := \lceil t \cdot \log_2 n \rfloor$ where t is a small constant, which is consistent with the complexity of the BKW algorithm $q^{\mathcal{O}(n/\log_2(n))} = 2^{\mathcal{O}(n)}$ if $q \in \text{poly}(n)$.

4.1 Regev's Original Parameters

In [29] Regev proposes a simple public-key encryption scheme with the suggested parameters $q \approx n^2$ and $\alpha = 1/(\sqrt{n} \cdot \log_2^2 n\sqrt{2\pi})$. We consider the parameters in the range $n = 32, \ldots, 256$. In our experiments $t = 3.0$ produced the best results, i.e., higher values of t resulted in m growing too fast. Plugging these values into the formulas of Theorem 2 we get an overall complexity of

$$\frac{mn^9 + \frac{1}{6}\,2^{\frac{2}{3}n}n^5 + \left[\left(3\,n + \frac{9}{2}\right) \cdot \left(2^{\frac{2}{3}n}n^4 + 1\right)\right] \log_2(n)^2 + \frac{1}{6}\,n}{2\,(n^4 - 1)}$$

operations in $\mathbb{Z}_q$ after simplification. If $m < 2^{(\frac{2}{2.6}n)}$) then this expression is dominated by

$$\frac{\frac{1}{6}n^5 + \left(3\,n^5 + \frac{9}{2}n^4\right) \cdot \log_2(n)^2}{2\,(n^4 - 1)} 2^{\frac{2}{3}n} \in 2^{\frac{2}{3}n + \mathcal{O}(\log n)}.$$

However, since we compute m numerically, we have to rely concrete values for various n to verify that with these settings indeed m does not grow too fast. Table 1 lists the estimated number of calls to $L_{\mathbf{s},\chi}$ ("$\log_2 \#L_{\mathbf{s},\chi}$"), the estimated number of required ring ("$\log_2 \#\mathbb{Z}_q$") and bit ("$\log_2 \#\mathbb{Z}_2$") operations, the costs in terms of ring operations for each of the three stages sampling, hypothesis testing and back substitution.

n	$\log_2 m$	$\log_2 \#\mathbb{Z}_q$ in				$\log_2 \#\mathbb{Z}_2$	$\log_2 \#L_{\mathbf{s},\chi}$
		sample	hypo.	subs.	total		
32	19.93	32.76	34.94	29.31	35.25	38.57	25.64
48	28.90	43.70	45.66	40.69	46.02	49.50	35.81
64	34.22	54.36	52.22	51.86	54.85	58.43	45.87
80	42.19	65.50	61.16	62.94	65.78	69.44	56.60
96	49.83	76.52	69.58	73.91	76.75	80.47	67.31
112	58.79	87.51	79.22	84.84	87.72	91.49	78.02
128	67.44	98.46	88.44	95.75	98.67	102.48	88.74
144	76.40	109.35	97.91	106.61	109.56	113.40	99.43
160	86.37	120.23	108.34	117.46	120.43	124.30	110.12
176	97.34	131.09	119.71	128.29	131.28	135.18	120.82
192	106.30	141.93	129.06	139.10	142.12	146.04	131.51
208	117.27	152.76	140.37	149.91	152.95	156.89	142.20
224	128.56	163.57	151.98	160.70	163.76	167.72	152.88
240	139.52	174.37	163.24	171.48	174.56	178.54	163.57
256	150.49	185.17	174.49	182.26	185.35	189.35	174.25

Table 1. Cost of solving Search-LWE for parameters suggested in [29] with $d = 1, t = 3, \epsilon = 0.99$ with BKW.

4.2 Lindner and Peikert's Parameters

In [21], Lindner and Peikert propose new attacks and parameters for LWE. Table 2 lists concrete costs of the BKW algorithm for solving LWE under the parameter choices from [21] as interpreted in [6]. In our computations $t = 2.7$ produced the best results, i.e., higher values of t resulted in m growing too fast.

n	$\log_2 m$	$\log_2 \#\mathbb{Z}_q$ in				$\log_2 \#\mathbb{Z}_2$	$\log_2 \#L_{\mathbf{s},\chi}$
		sample	hypo.	subs.	total		
32	7.64	35.91	33.65	33.92	36.46	39.78	28.84
48	9.97	45.75	36.56	43.58	46.04	49.52	37.94
64	15.61	54.82	42.62	52.53	55.09	58.67	46.49
80	22.25	63.39	49.58	61.02	63.65	67.31	54.66
96	30.90	71.62	58.49	69.18	71.86	75.58	62.57
112	40.86	79.57	68.68	77.08	79.81	83.58	70.25
128	54.15	87.32	82.16	84.78	87.58	91.39	77.76
144	37.54	102.31	67.71	99.73	102.53	106.37	92.54
160	46.51	110.38	76.83	107.77	110.60	114.47	100.43
176	56.47	118.31	86.93	115.68	118.53	122.43	108.20
192	70.76	126.13	101.35	123.47	126.34	130.26	115.87
208	80.73	133.84	111.43	131.15	134.05	137.99	123.44
224	94.34	141.45	125.15	138.74	141.66	145.62	130.92
240	109.62	148.98	140.53	146.25	149.18	153.16	138.33
256	126.23	156.42	157.24	153.68	157.96	161.96	145.67

Table 2. Cost of solving Search-LWE for parameters suggested in [21] with $d = 1, t = 2.7, \epsilon = 0.99$ with BKW.

4.3 Albrecht et al.'s Polly-Cracker

In [5] a somewhat homomorphic encryption scheme is proposed based on the hardness of computing Gröbner bases with noise. Using linearisation the equation systems considered in [5] may be considered as LWE instances. Table 3 lists concrete costs for recovering the secret Gröbner basis using this strategy for selected parameters suggested in [5]. In Table 3 "λ" is the targeted bit-security level and n the number of variables in the linearised system. We note that we did not exploit the structure of the secret for Table 3.

λ	n	q	α	t	$\log_2 m$	$\log_2 \#\mathbb{Z}_q$ in				$\log_2 \#\mathbb{Z}_2$	$\log_2 \#L_{\mathbf{s},\chi}$
						sample	hypo.	subs.	total		
80	136	1999	0.005582542...	2.2	93.58	109.40	121.59	105.71	121.59	125.04	100.23
	231	92893	0.000139563...	3.4	127.23	157.47	167.09	154.40	167.09	171.13	146.54
128	153	12227	0.002797408...	2.4	84.05	132.07	117.45	129.66	132.32	136.08	122.39
	253	594397	0.000034967...	3.8	100.66	175.15	146.00	171.88	175.29	179.55	163.89

Table 3. Cost of finding $G \approx \mathbf{s}$ for parameters suggested in [5] with $d = 2, \epsilon = 0.99$.

5 Comparison with Alternative Approaches

Now, given the complexity estimates in Section 4 we may ask how these relate to existing approaches in the literature. Hence, we briefly describe some alternative strategies for solving the LWE problem.

5.1 Short Integer Solutions: Lattice Reduction

In [24], the authors briefly examine an approach for solving LWE by distinguishing between valid matrix-LWE samples of the form $(\mathbf{A}, \mathbf{c}) = (\mathbf{A}, \mathbf{As} + \mathbf{e})$ and samples drawn from the uniform distribution over $\mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^n$. Given a matrix of samples $\mathbf{A}$, one way of constructing such a distinguisher is to find a short vector $\mathbf{u}$ in the dual lattice $\Lambda(\mathbf{A})^\perp$ such that $\mathbf{uA} = \mathbf{0} \mod q$. If $\mathbf{c}$ belongs to the uniform distribution over $\mathbb{Z}_q^n$, then $\langle \mathbf{u}, \mathbf{c} \rangle$ belongs to the uniform distribution on $\mathbb{Z}_q$. On the other hand, if $\mathbf{c} = \mathbf{As} + \mathbf{e}$, then $\langle \mathbf{u}, \mathbf{c} \rangle = \langle \mathbf{u}, \mathbf{As} + \mathbf{e} \rangle = \langle \mathbf{u}, \mathbf{e} \rangle$, where samples of the form $\langle \mathbf{u}, \mathbf{e}_i \rangle$ are governed by another discrete, wrapped Gaussian distribution. Following the work of Micciancio and Regev [24], the authors of [21] give estimates for the complexity of distinguishing between LWE samples and uniform samples by estimating the cost of the BKZ algorithm in finding a short enough vector. In particular, given $n, q, \sigma = \alpha q$, We set $s = \sigma \cdot \sqrt{2\pi}$ and compute $\beta = q/s \cdot \sqrt{\log(1/\epsilon)/\pi}$. From this β we then compute the required root Hermite factor $\delta = 2^{\log_2^2(\beta)/(4n \log_2 q)}$. Note that this presupposes access to $m = \sqrt{n \log_2 q / \log_2 \delta}$ samples; an assumption which holds in our setting. Given δ we then extrapolate the running time in seconds as $\log_2 T_{sec} = 1.8/\log_2 \delta - 110$ as in [21]. We translate this figure into bit operations by assuming $2.3 \cdot 10^9$ bit operations per second on a 2.3 GHz CPU. Furthermore, we note that for BKZ picking $\epsilon \ll 1$ and running the algorithms about $1/\epsilon$ times is usually more efficient than picking $\epsilon \approx 1$ directly. Table 4 compares the number of bit and ring operations using the BKW and BKZ algorithm as described in [21]. In Table 4 running times and the number of required samples for BKZ include the $1/\epsilon$ factor, hence both approaches distinguish with probability close to 1.

Hence, Table 4 illustrates that for the families of parameters considered here, we expect the BKW algorithm to be asymptotically faster than the BKZ algorithm with a crossover around $n = 250$ at the cost of requiring a lot more samples and memory.

n	q	αq	BKW					NTL-BKZ Lindner/Peikert Model				
			t	$\log_2 m$	$\log_2 \#\mathbb{Z}_q$	$\log_2 \#\mathbb{Z}_2$	$\log_2 \#L_{\mathbf{s},\chi}$	$\log_2 \epsilon$	$\log_2 m$	$\log_2 \#\mathbb{Z}_q$	$\log_2 \#\mathbb{Z}_2$	$\log_2 \#L_{\mathbf{s},\chi}$
Regev [29]												
128	16411	11.81	3.2	81.62	93.84	97.65	83.85	-18	26.47	61.56	65.36	26.47
256	65537	25.53	3.1	126.26	179.76	183.76	168.79	-29	38.50	175.48	179.48	38.50
512	262147	57.06	3.1	337.92	350.80	354.97	338.02	-48	58.52	386.75	390.92	58.52
Lindner & Peikert [21]												
128	2053	2.70	2.9	63.86	82.40	85.86	72.73	-18	26.25	54.50	57.96	26.25
256	4099	3.34	2.8	105.08	151.45	155.04	140.64	-29	38.22	156.18	159.77	38.22
512	4099	2.90	2.6	157.78	278.01	281.59	266.14	-50	60.14	341.87	345.45	60.14

Table 4. Cost of solving Decision-LWE with BKZ as in [21] and BKW as in Corollary 3.

In [12] the authors present a study of 'BKZ 2.0', the amalgamation of three folklore techniques to improve the performance of BKZ: pruned enumeration; pre-processing of local blocks and early termination. While no implementations of such algorithms are publicly available, the authors of [12] present a simulator to predict the behaviour of out-of-reach BKZ algorithms. However, the accuracy of this simulator has not been independently verified. In a recent work [22], the authors re-visit the BKZ running-time model of [21] and compare the predictions to the simulator of [12] in a few cases. In the cases examined in [22], the running-time predictions obtained by the use of the BKZ 2.0 simulator are quite close to those obtained by the model of Lindner and Peikert.

Based on the data-points provided in [22] and converting these to the same metric as in the Lindner-Peikert model, the function

$$\log_2 T_{sec}^{\text{BKZ2.0}} = 0.009/\log_2^2 \delta_0 - 27$$

provides a very close approximation to the running-time output of the simulator for this particular case (cf. Figure 1).

This is a non-linear approximation and hence naturally grows faster than the approximation in [21]. This does not imply that the BKZ 2.0 algorithm is slower than the variant implemented in NTL, as these are two different estimates for the same algorithm (the extrapolation of [21] aimed to take into account the advances collectively known as BKZ 2.0). However, given the greater sophistication of the latter "BKZ 2" extrapolations derived from the simulator of [12], we expect this model to provide more accurate approximations of running times than the model of [21].

In particular, a BKZ logarithmic running-time model which is non-linear in $\log_2(\delta_0)$ appears more intuitive than a linear model. While, in practise, the root Hermite factors achievable through the use of BKZ with a particular blocksize β are much better than their best provable upper bounds, the root factor achievable appears to behave similarly to the upper bounds as a function of β. Namely, the best proven upper bounds on the root Hermite factor are of the form $\sqrt{\gamma_\beta}^{1/(\beta-1)}$, where γ_β denotes the best known upper bound on the Hermite constant for lattices of dimension β. Now since, asymptotically, γ_β grows linearly in β, if we assume that the root Hermite factor achievable in practise displays asymptotic behaviour similar to that of the best-known upper bound, then the root Hermite factor achievable as a function of β, denoted $\delta_0(\beta)$, is such that $\delta_0(\beta) \in \Omega(1/\beta)$. Since the running time of BKZ appears to be doubly-exponential in β, we can derive that $\log T_{sec}$ is non-linear in $1/\log(\delta_0)$, as is borne out by the results in [22]. We also note that in [21] the assumption is made that $\log T_{sec} = \mathcal{O}(1/\log(\delta_0))$, which does not hold from the above discussion.

Using this model, we can give an analogue of Table 4 in which the BKZ entries are obtained using the above approximate model.

Thus, we can reasonably conclude that, under the assumptions made above, employing lattice reduction in a pure distinguishing approach is out-performed by BKW in surprisingly low dimension.

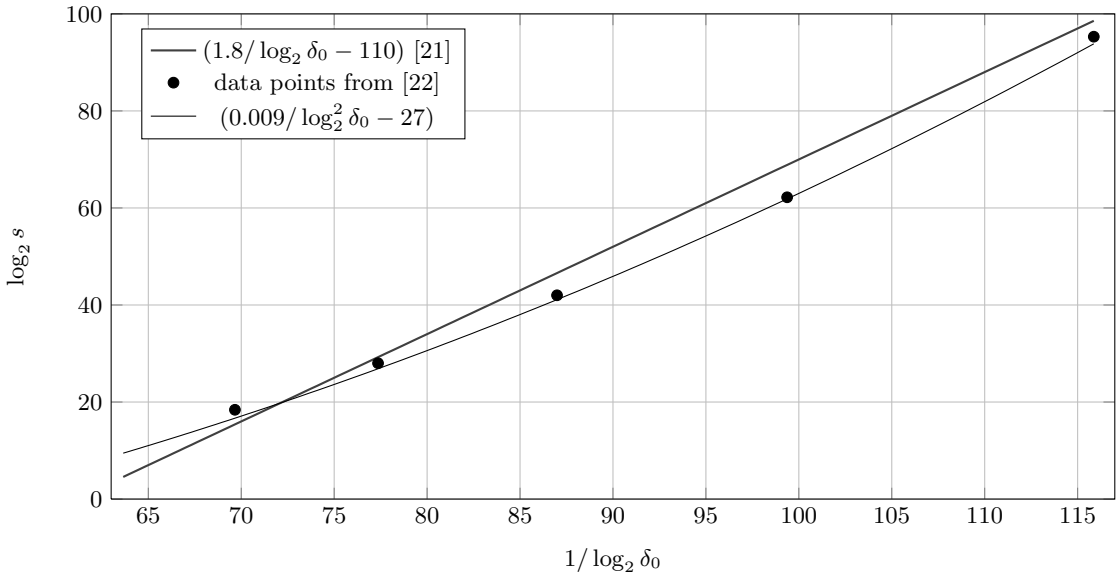

Fig. 1. BKZ running times in seconds s for given values of δ_0.

n	q	αq	BKW					BKZ 2.0 Simulator Model				
			t	$\log_2 m$	$\log_2 \#\mathbb{Z}_q$	$\log_2 \#\mathbb{Z}_2$	$\log_2 \#L_{\mathbf{s},\chi}$	$\log_2 \epsilon$	$\log_2 m$	$\log_2 \#\mathbb{Z}_q$	$\log_2 \#\mathbb{Z}_2$	$\log_2 \#L_{\mathbf{s},\chi}$
Regev [29]												
128	16411	11.81	3.2	81.62	93.84	97.65	83.85	-14	22.50	61.90	65.71	22.50
256	65537	25.53	3.1	126.26	179.76	183.76	168.79	-35	44.48	174.46	178.46	44.48
512	262147	57.06	3.1	337.92	350.80	354.97	338.02	-94	104.47	518.62	522.79	104.47
Lindner & Peikert [21]												
128	2053	2.70	2.9	63.86	82.40	85.86	72.73	-14	22.28	57.06	60.52	22.28
256	4099	3.34	2.8	105.08	151.45	155.04	140.64	-33	42.21	151.16	154.74	42.21
512	4099	2.90	2.6	157.78	278.01	281.59	266.14	-86	96.09	424.45	428.03	96.09

Table 5. Cost of distinguishing LWE samples from uniform as reported as "Distinguish" in [21], compared to Corollary 3. BKZ estimates obtained using BKZ 2.0 simulator-derived cost estimate.

5.2 Short Integer Solutions: Combinatorial

Recall that if we consider the set of samples from $L_{\mathbf{s},\chi}$ used during the course of the BKW algorithm as determining a q-ary lattice, and the noisy vector as denoting a point close to a lattice point, we may consider the BKW algorithm as analysed in Corollary 3 as a combinatorial approach for sampling a sparse $\mathbf{u}$ in the dual lattice with entries $\in \{-1, 0, 1\}$. Hence, it is related to a combinatorial approach for finding short dual-(q-ary)lattice vectors as briefly sketched in [24, p. 156] (cf. also [23]). These algorithms, however, operate somewhat differently to BKW - given a relatively small set of LWE samples, these are divided into a small number of subsets. Within each subset, we compute all linear combinations of the members of that subset such that the coefficients of these linear combinations are in $\{-b', \ldots, b'\}$ (note that the parameter b' is unrelated to the parameter b used in this work).

The algorithm sketched by [24] uses the generalised birthday paradox to produce collisions among samples produced by inter-addition, with the parameters of the algorithm being chosen such that we expect to obtain a single short vector in the dual-lattice vector.

There are some significant differences between such algorithms and BKW, stemming from the assumption of the former that we only have access to a very small number of LWE samples. This requires the 'expansion' of the sample set in such a way that when we search for collisions, we are almost certain to find enough. On the other hand, due to this expansion, the initial samples must be separated into disjoint lists. Thus, probably the easiest way to describe the algorithm in [24] in terms of BKW is to imagine a variant of BKW where, if a sample is not in a given table, we add this sample plus *all* $\{-b, \ldots, b\}$-bounded linear combinations of this sample with the pre-existing table entries, to the table. To give a strict analogue to [24], on finding a subsequent collision, we would need to store a list of all noise elements which had been added to a sample and ensure that, if we want to eliminate a sample, that the set of noise elements belonging to the sample and the set of noise elements belonging to the table entry are disjoint.

However, the fundamental difference stems from the assumption that the number of samples is restricted. If this is not the case (as we assume), then it is clear that BKW delivers a much shorter dual lattice vector.

5.3 Bounded Distance Decoding: Lattice Reduction

In [21], the authors propose a method to solve LWE instances which consists of q-ary lattice reduction, then employing a decoding stage to determine the secret. The decoding stage used is essentially a straightforward modification of Babai's well-known nearest plane algorithm for CVP. The authors estimate the running-time of the BKZ algorithm in producing a basis 'reduced-enough' for the decoding stage of the algorithm to succeed, then add the cost of the decoding stage.

To obtain comparable complexity results, we calculate upper bounds on the bit operation counts for two data-points based on the running times reported in [21] multiplied by the clock speed of the CPU used. As can be seen from Table 6, unsurprisingly these indicate substantially lower complexities than for BKW. In addition, the memory requirements of this approach are small compared to the memory requirements of BKW.

n	q	αq	BKW					NTL-BKZ Lindner/Peikert Model				
			t	$\log_2 m$	$\log_2 \#\mathbb{Z}_q$	$\log_2 \#\mathbb{Z}_2$	$\log_2 \#L_{\mathbf{s},\chi}$	$\log_2 \epsilon$	$\log_2 m$	$\log_2 \#\mathbb{Z}_q$	$\log_2 \#\mathbb{Z}_2$	$\log_2 \#L_{\mathbf{s},\chi}$
136	2003	5.19	2.6	67.49	93.77	97.23	84.15	-25	33.46	91.35	94.81	33.46
214	16381	2.94	3.4	76.90	128.36	132.16	117.54	-18	26.95	82.31	86.11	26.95

Table 6. Cost of solving Search-LWE reported as "Decode" in [21], compared to the cost solving Decision-LWE with BKW

5.4 Bounded Distance Decoding: Combinatorial

We can take the approach of viewing LWE as being the problem of solving BDD in a random q-ary lattice with a random target $\mathbf{t} = \mathbf{As} + \mathbf{e}$. To solve the LWE problem formulated thus, we have several choices of lattice-based algorithms. However, for several such algorithms, tight complexity estimates are unavailable, thus making comparison to BKW loose. One algorithm for which more precise complexity estimates are available is the AKS (Ajtai, Kumar & Sivakumar) algorithm for solving the shortest vector problem [2]. This algorithm is notable as being the first proposed singly-exponential SVP algorithm.

It is beyond the scope of this paper to give details of the AKS algorithm – we are interested solely in the complexity. In the original paper by Ajtai, Kumar & Sivakumar, no explicit running-time bound was given.

Subsequent developments of the AKS algorithm by Regev, Nguyen and Vidick, Micciancio and Voulgaris, Pujol and Stehle delivered algorithms which were of time complexity $2^{16n+o(n)}$, $2^{5.9n+o(n)}$, $2^{3.4n+o(n)}$ and $2^{2.7n+o(n)}$, respectively [18].

6 Experimental Results

In order to verify the results of this work, we implemented stages 1 and 2 of the BKW algorithm. Our implementation considers LWE with short secrets but we ignore the transformation cost to produce samples with a short secret. Also, our implementation supports arbitrary bit-width windows b, not only multiplies of $\lceil \log_2(q) \rceil$. However, due to the fact that our implementation does not use a balanced representation of finite field elements internally – which simplifies dealing with arbitrary bit-width windows – our implementation does not *fully* implement the half-table improvement. That is, for simplicity, our implementation only uses the additive inverse of a vector if this is trivially compatible with our internal data representation. Furthermore, our implementation does not bit-pack finite field elements. Elements always take up 16 bits of storage. Overall, the memory consumption of our implementation in stage 1 is worse by a factor of up to four compared to the estimates given in this work and the computational work in stage is worse by a factor of up to two. Finally, since our implementation is not optimised we do not report CPU times.

With these considerations in mind, our estimates are confirmed by our implementation. For example, consider Regev's parameters for $n = 25$ and $t = 2.3$ and $d = 1$, By Lemma 9 picking $m = 2^{12.82}$ will result in a success probability of $\mathrm{p}_{success} \approx 0.99959$ per component and $\mathrm{P}_{success} \approx 0.99$ overall. Lemma 2 estimates a computational cost of $2^{30.54}$ ring operation and $2^{24.19}$ calls to $L_{\mathbf{s},\chi}$ in stage 1. We ran our implementation with $m = \lceil 2^{12.82} \rfloor$ and window bitsize $w = 22 = \frac{n \log_2(q)}{2.3 \log_2(n)}$. It required $2^{29.74}$ ring operations and $2^{23.31}$ calls to $L_{\mathbf{s},\chi}$ to recover one component of $\mathbf{s}$. From this we conclude that Theorem 2 is reasonably tight.

To test the accuracy of Lemma 9 we ran our implementation with the parameters $n = 25$, $q = 631$, $\alpha \cdot q = 5.85$, $w = 24 = \frac{n \log_2(q)}{2.1 \log_2(n)}$ and $m = 2^7$. Lemma 9 predicts a success rate of 53%. In 1000 experiments we 665 times rank zero for the correct key component, while Lemma 9 predicted 530. Hence, it seems our predictions are slightly pessimistic. The distribution of the ranks of the correct component of $\mathbf{s}$ in 1000 experiments is plotted in Figure 2.

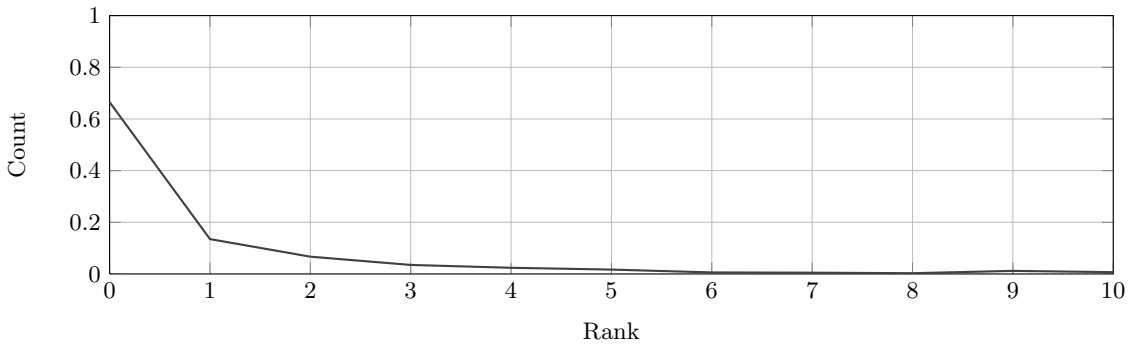

Fig. 2. Distribution of right key component ranks for 1000 experiments on $n = 25$, $t = 2.3$, $d = 1$, $\mathrm{p}_{success} = 0.99$.

Our implementation is available at `http://bitbucket.org/malb/bkw-lwe`.

7 Conclusion and Further Work

In this work we have provided a concrete analysis of the cost of running the BKW algorithm on LWE instances both for the search and the decision variants of the LWE problem. We also applied this analysis to various sets of parameters found in the literature. From this we conclude that the BKW algorithm outperforms lattice reduction algorithms in an SIS setting for the parameter sets proposed in [29, 21] starting around dimension $n \approx 250$ at the cost of requiring many more samples and storage. On the other hand, lattice reduction in a BDD setting currently outperforms the BKW algorithm as analysed in this work.

A pressing research question for future work is hence how to apply a variant of the BKW algorithm to the BDD problem. Furthermore, in this work we ignore the so-called LWE "normal form" where the secret follows the noise distribution χ (cf. [10]) and other small secret variants of LWE. For the BKW algorithm as presented in this work, only hypothesis testing is affected by the size of the secret and hence we do not expect the algorithm to benefit from considering small secrets. Yet, a dedicated variant of the BKW algorithm tackling small secrets is a promising research direction.

Acknowledgements

We are grateful to Frederik Johansson for advice on numerical integration. We are also grateful to anonymous referees whose feedback substantially improved this work. The work described in this paper has been partially supported by the Royal Society grant JP090728 and by the Commission of the European Communities through the ICT program under contract ICT-2007-216676 (ECRYPT-II). Jean-Charles Faugère, and Ludovic Perret are also supported by the Computer Algebra and Cryptography (CAC) project (ANR-09-JCJCJ-0064-01) and the HPAC grant of the French National Research Agency. Carlos Cid is supported in part by the US Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

1. Shweta Agrawal, Craig Gentry, Shai Halevi, and Amit Sahai. Discrete Gaussian Leftover Hash Lemma over infinite domains. Cryptology ePrint Archive, Report 2012/714, 2012. `http://eprint.iacr.org/`.
2. Miklós Ajtai, Ravi Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. In *IEEE Conference on Computational Complexity*, pages 53–57, 2002.
3. Martin Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the Arora-Ge algorithm against LWE. In *SCC '12: Proceedings of the 3nd International Conference on Symbolic Computation and Cryptography*, pages 93–99, Castro-Urdiales, July 2012.
4. Martin R. Albrecht. `bkw-estimator.py`, 2012. available at `https://bitbucket.org/malb/research-snippets/`.
5. Martin R. Albrecht, Pooya Farshim, Jean-Charles Faugère, and Ludovic Perret. Polly Cracker, revisited. In *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 179–196, Berlin, Heidelberg, New York, 2011. Springer Verlag. full version available as Cryptology ePrint Archive, Report 2011/289, 2011 `http://eprint.iacr.org/`.
6. Martin R. Albrecht, Robert Fitzpatrick, Daniel Cabracas, Florian Göpfert, and Michael Schneider. A generator for LWE and Ring-LWE instances, 2013. available at `http://www.iacr.org/news/files/2013-04-29lwe-generator.pdf`.
7. Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415, Berlin, Heidelberg, New York, 2011. Springer Verlag.
8. Thomas Baigneres, Pascal Junod, and Serge Vaudenay. How far can we go beyond Linear Cryptanalysis? In Pil Joong Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 432–450, Berlin, Heidelberg, New York, 2004. Springer Verlag.
9. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.
10. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of Learning with Errors. to appear STOC 2013, 2013.
11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 97–106. IEEE, 2011.
12. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: better lattice security estimates. In *Advances in Cryptology - ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20, Berlin, Heidelberg, 2011. Springer Verlag.
13. Lutz Duembgen. Bounding standard gaussian tail probabilities. *arXiv:1012.2063*, 2010.
14. Pierre-Alain Fouque and Éric Levieil. An improved LPN algorithm. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks, 5th International Conference, SCN 2006*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer Verlag, 2006.
15. Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 257–278. Springer Verlag, 2010.
16. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. Available at `http://crypto.stanford.edu/craig`.
17. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC 08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.
18. Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *IWCC*, volume 6639 of *Lecture Notes in Computer Science*, pages 159–190. Springer, 2011.
19. Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 447–464. Springer Verlag, 2011.
20. Fredrik Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.17)*, February 2011. `http://code.google.com/p/mpmath/`.
21. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339, Berlin, Heidelberg, New York, 2011. Springer Verlag.

22. Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In Ed Dawson, editor, *CT-RSA*, volume 7779 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2013.
23. Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In Kaisa Nyberg, editor, *Fast Software Encryption*, volume 5086 of *Lecture Notes in Computer Science*, pages 54–72. Springer Verlag, 2008.
24. Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer Verlag, Berlin, Heidelberg, New York, 2009.
25. Ivan Morel, Damien Stehlé, and Gilles Villard. H-LLL: using householder inside LLL. In Jeremy R. Johnson, Hyungju Park, and Erich Kaltofen, editors, *Symbolic and Algebraic Computation, International Symposium, ISSAC 2009*, pages 271–278. ACM, 2009.
26. Phong Q. Nguyen. Lattice reduction algorithms: Theory and practice. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 2–6. Springer Verlag, 2011.
27. Phong Q. Nguyen and Damien Stehlé. Low-dimensional lattice basis reduction revisited. *ACM Transactions on Algorithms*, 5(4), 2009.
28. Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *IACR Cryptology ePrint Archive*, 2009:605, 2009.
29. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.
30. Oded Regev. The learning with errors problem (invited survey). In *IEEE Conference on Computational Complexity*, pages 191–204. IEEE Computer Society, 2010.
31. Markus Rückert and Michael Schneider. Estimating the security of lattice-based cryptosystems. *IACR Cryptology ePrint Archive*, 2010:137, 2010.
32. W. A. Stein et al. *Sage Mathematics Software (Version 5.2)*. The Sage Development Team, 2012. `http://www.sagemath.org`.