Lazy Modulus Switching for the BKW Algorithm on LWE

Martin R. Albrecht[1], Jean-Charles Faugère[3,2,4], Robert Fitzpatrick[5], and Ludovic Perret[2,3,4]

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

Abstract. Some recent constructions based on LWE do not sample the secret uniformly at random but rather from some distribution which produces small entries. The most prominent of these is the binary-LWE problem where the secret vector is sampled from $\{0,1\}^*$ or $\{-1,0,1\}^*$. We present a variant of the BKW algorithm for binary-LWE and other small secret variants and show that this variant reduces the complexity for solving binary-LWE. We also give estimates for the cost of solving binary-LWE instances in this setting and demonstrate the advantage of this BKW variant over standard BKW and lattice reduction techniques applied to the SIS problem. Our variant can be seen as a combination of the BKW algorithm with a lazy variant of modulus switching which might be of independent interest.

1 Introduction

Learning With Errors (LWE) [22] has received widespread attention from the cryptographic community since its introduction. LWE-based cryptography is mainly motivated by its great flexibility for instantiating cryptographic solution as well as a deep worst-case/average-case connections [22]: solving LWE on the average is not easier than solving worst-case instances of several famous lattice approximation problems.

The motivation behind this work comes from the observation that some recent constructions based on LWE do not sample the secret uniformly at random but rather from some distribution which produces small entries (e.g. [5,1,13,12,21]). From a theoretical point of view, this is motivated by the observation that every LWE instance can be transformed into an instance where the secret follows the same distribution as the noise [5].[1] However, many constructions use secrets which are considerably smaller. For example, binary-LWE samples the secret from $\{0,1\}^*$ [8] or $\{-1,0,1\}^*$ [12]. The presence of such small secrets provokes the question of what implications such choices have on the security of LWE. Is solving LWE with, say, binary secrets easier than standard LWE? From a theoretical point of view, [8] proves that their binary-LWE is as secure as LWE. In this paper, we try to address the question from an algorithmic point of view; i.e. what is the actual impact of small secrets on concrete parameters.

[1] also in [16] for the LPN case.

1.1 Algorithms for Solving LWE

Three families of algorithms for solving LWE are known in the literature. The most prominent approach is to reduce LWE to a problem that can be solved via lattice reduction, for example, by reducing it to the Short Integer Solution (SIS) problem. Indeed, most parameter choices in the literature are based on the hardness of lattice reduction such as [17,10,18]. These estimates for a given set of parameters n (number of components of the secret), q (size of the modulus) and σ (standard deviation of the noise) are usually produced by extrapolating running times from small instances.

A second approach is due to Arora and Ge who reduce LWE to solving a system of non-linear equations [6]. This algorithm allow us to solve LWE in sub-exponential time as soon as the Gaussian distribution is sufficiently narrow, i.e. $\alpha \cdot q < \sqrt{n}$. Recall that the security reduction [22] for LWE requires to consider discrete Gaussian with standard deviation $\alpha \cdot q$ strictly bigger than $\sqrt{n}$. However, from a practical point of view, the constants involved in this algorithm are so large that it is much more costly than other approaches for the parameters typically considered in cryptographic applications [2].

The third family of algorithms are combinatorial algorithms which can all be seen as variants of the BKW algorithm. The BKW algorithm was proposed by Blum, Kalai and Wasserman [7] as a method for solving the Learning Parity with Noise problem, with sub-exponential complexity, requiring $2^{\mathcal{O}(n/\log n)}$ samples, space and time. The algorithm can be adapted for tackling LWE with complexity $2^{\mathcal{O}(n)}$ when the modulus is taken to be polynomial in n [22]. BKW proceeds by splitting the n components of LWE samples into a groups of b components each. For each of the a groups of components the algorithm then searches for collisions in these b components to eliminate them. The overall complexity of the algorithm is $\approx \left(a^2 n\right) \cdot \frac{q^b}{2}$ operations, and $a \cdot \frac{q^b}{2}$ memory, where a and b depend on the n, q and α.

The behaviour of the algorithm is relatively well understood and it was shown to outperform lattice reduction estimates when reducing LWE to SIS (when q is small), thus it provides a solid basis for analysing the concrete hardness of LWE instances [3].

1.2 Organisation of the Paper and Main Results

While none of the algorithms above take advantage of the presence of small secrets, we may combine them with *modulus switching*. Recall that modulus switching was initially introduced to improve the performance of homomorphic encryption schemes [9] and was recently used to reduce the hardness of LWE with polynomially sized moduli to GAPSVP [8]. Modulus switching is essentially the same as computing with a lower precision similar to performing floating point computations with a low fixed precision. Namely, let $\left(\mathbf{a}, c = \langle \mathbf{a}, \mathbf{s} \rangle + e\right) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ be LWE sample where $\mathbf{s} \in \mathbb{Z}_q^n$ is the secret vector, and $e \in \mathbb{Z}_q$ is an error. Let also some $p < q$ and consider $\left(\lfloor p/q \cdot \mathbf{a} \rceil, \lfloor p/q \cdot c \rceil \right)$ with

$$
\begin{aligned}
\left\lfloor \frac{p}{q} \cdot c \right\rceil &= \left\lfloor \frac{p}{q} \big(\langle \mathbf{a}, \mathbf{s} \rangle + q \cdot u + e \big) \right\rceil, \text{ for some } u \in \mathbb{Z} \\
\left\lfloor \frac{p}{q} \cdot c \right\rceil &= \left\lfloor \left\langle \frac{p}{q} \cdot \mathbf{a}, \mathbf{s} \right\rangle_p + \frac{p}{q} \cdot e \right\rceil = \left\lfloor \left\langle \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + \left\langle \frac{p}{q} \cdot \mathbf{a} - \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + \frac{p}{q} \cdot e \right\rceil \\
&= \left\langle \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + \left\langle \frac{p}{q} \cdot \mathbf{a} - \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + \frac{p}{q} \cdot e + e', \text{ where } e' \in [-0.5, 0.5] \\
&= \left\langle \left\lfloor \frac{p}{q} \cdot \mathbf{a} \right\rceil, \mathbf{s} \right\rangle_p + e'' + \frac{p}{q} \cdot e + e'. \qquad (1)
\end{aligned}
$$

where $\langle \mathbf{x}, \mathbf{y} \rangle_p$ denotes the modulo p inner product of $\mathbf{x}$ and $\mathbf{y}$.

Since $p/q \cdot \mathbf{a} - \lfloor p/q \cdot \mathbf{a} \rceil$ takes values $\in [-0.5, 0.5]$ we have that e'' is small if $\mathbf{s}$ is small. We may hence compute with the smaller 'precision' p at the cost of a slight increase of the noise rate by a '*rounding error*' e''.

Modulus switching allows to map a LWE instance $\bmod q$ to a scaled instance of LWE $\bmod p$. Thus, modulus switching can be used in the solving of small secret instances of LWE, a folklore approach which has not been explicitly studied in the literature. Namely, if we pick p such that e'' is not much larger than $p/q \cdot e$ then, for example, the running time of the BKW algorithm improves from $(a^2 n) \cdot \frac{q^b}{2}$ to $(a^2 n) \cdot \frac{p^b}{2}$. Since typically $b \approx n/\log n$ this may translate to substantial improvements. Indeed, we can pick p such that $|\langle p/q \cdot \mathbf{a} - \lfloor p/q \cdot \mathbf{a} \rceil, \mathbf{s} \rangle| \approx p/q \cdot |e|$. This implies $\sigma_s \cdot \sqrt{\frac{n}{12}} \approx p/q \cdot \sigma$, or $p \approx \min\left\{q, \frac{\sigma_s}{\sigma} \cdot \sqrt{\frac{n}{12}} \cdot q\right\}$, where σ_s is the standard deviation of elements in the secret $\mathbf{s}$.

In this paper, we refine this approach and present a variant of the BKW algorithm which fuses modulus switching and BKW-style reduction. In particular, this work has two main contributions. Firstly, in Section 2 we present a modulus switching strategy for the BKW algorithm in which switching is delayed until necessary. In a nutshell, recall that the BKW algorithm performs additions of elements which collide in certain components. Our variant will search for such collisions in 'low precision' $\mathbb{Z}_p$ but will perform arithmetic in 'high precision' $\mathbb{Z}_q$. We call *rounding error* the inner product of the sub-vector of 'low bits' of $\mathbf{a}$ with the secret $\mathbf{s}$. Our strategy permits to decrease rounding errors and allows to reduce p by a factor of $\sqrt{a}$.

Secondly, this perspective enables us to choose reductors in the BKW algorithm which minimise the rounding errors further (Section 3). Namely, we favour components a with small distance $|\lfloor p/q \cdot a \rceil - p/q \cdot a|$ in already reduced components, called 'child components' in this work. Our strategy ensures that the probability of finding such elements is highest for those components which are considered first by the BKW algorithm, i.e. those components which contribute most to the noise. We note that the first contribution relies on standard independence assumptions only, while the second contribution relies on stronger assumptions, which however seem to hold in practice.

We then discuss the complexity of our variants in Section 4. For typical choices of parameters – i.e. $q \approx n^c$ for some small constant $c \geq 1$, $a = \log_2 n$ and $b = n/\log_2 n$ – the complexity of BKW as analysed in [3] is $\mathcal{O}\left(2^{cn} \cdot n \log_2^2 n\right)$. For small secrets, a naive modulus switching technique allows reducing this complexity to $\mathcal{O}\left(2^{n\left(c + \frac{\log_2 d}{\log_2 n}\right)} \cdot n \log_2^2 n\right)$ where $0 < d \leq 1$ is a small constant. If the secret distribution does not depend on n and if an unbounded number of LWE samples is available our improved version of BKW allows to get a complexity of:

$$\mathcal{O}\left(2^{n\left(c + \frac{\log_2 d - \frac{1}{2}\log_2 \log_2 n}{\log_2 n}\right)} \cdot n \log_2^2 n\right).$$

We then study the behaviour of this algorithm by applying it to various instances of LWE with binary secrets. In Section 5 we report on experiments conducted with a proof-of-concept implementation of our algorithm. In Section 6, we compare the results with plain BKW and BKZ under modulus switching and a simple meet-in-the-middle approach or generalised birthday attack. We show that our lazy-modulus-switching variant of the BKW algorithm provides better results than applying plain BKW after modulus reduction. We also demonstrate that under the parameters considered here this algorithm also – as n increases – outperforms the most optimistic estimates for BKZ when we apply BKZ to the same task as that to which we apply BKW: finding short vectors in the (scaled-)dual lattice – we obtain this perspective by viewing the rounding error as an increase in the noise rate while still finding short vectors in the (scaled)-dual p-ary lattice determined by our modulus-reduced LWE samples. Indeed, our results indicate that our algorithm outperforms BKZ 2.0 when both are used to find a short vector in the (scaled)-dual lattice in dimension as low as ≈ 256 when considering LWE parameters from [22] with binary secret. How-

ever, we stress again that we always assume an unbounded number of samples to be available for solving.

1.3 Notations

To fix the notations, we reproduce below the definition of LWE.

Definition 1 (LWE [22]). *Let n, q be positive integers, χ be a probability distribution on $\mathbb{Z}_q$ 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 $e \in \mathbb{Z}_q$ according to χ, and returning $(\mathbf{a}, c) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. We define* Decision-LWE *as 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 $L_{\mathbf{s},\chi}$.*

The noise follows some distribution χ which is classically chosen to be a discrete Gaussian distribution over $\mathbb{Z}$ with mean 0 and standard deviation $\sigma = s/\sqrt{2\pi} = \alpha q/\sqrt{2\pi}$, reduced modulo q. In the following, we always start counting at zero. We denote vectors as well as matrices in bold, vectors in lower case, and matrices in upper case. 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 entry at index i, j. For vectors $\mathbf{a}$ we denote by $\mathbf{a}_{(a,b)}$ the vector $(\mathbf{a}_{(a)}, \ldots, \mathbf{a}_{(b-1)})$. 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. This means that $\mathbf{a}_{i,(j)}$ is the j-th component of the vector $\mathbf{a}_i$. 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 initial distribution. We write $\tilde{\mathbf{a}}$ instead of $\mathbf{a}$ to indicate $\mathbf{a}$ has some short elements. We represent elements in $\mathbb{Z}_q$ as integers in $[-\frac{q}{2}, \ldots, \frac{q}{2}]$, similarly for $\mathbb{Z}_p$. We write $\chi_{\alpha,q}$ for the distribution obtained by considering a discrete Gaussian distribution over $\mathbb{Z}$ with standard deviation $\alpha q/\sqrt{2\pi}$, mean 0, considered modulo q.

2 A Modified BKW Algorithm: Lazy Modulus Switching

Following [3], we consider BKW – applied to Decision-LWE – as consisting of two stages: *sample reduction* and *hypothesis testing*. In this work, we only modify the first stage.

2.1 The Basic Idea

We briefly recall the principle of classical BKW. Assume we are given samples of the form $(\mathbf{a}, c)$ following either $L_{\mathbf{s},\chi}$ or $\mathcal{U}(\mathbb{Z}_q^n) \times \mathcal{U}(\mathbb{Z}_q)$. Our goal is to distinguish between the two cases. BKW proceeds by producing samples $(\mathbf{a}^*, c^*)$ with $\mathbf{a}^*$ being all zero such that statistical tests can be applied to c^* to decide whether they follow $\mathcal{U}(\mathbb{Z}_q)$ or some distribution related to $L_{\mathbf{s},\chi}$. This is achieved by grouping the n components of all vectors into a groups of b components each (assuming a and b divide n for simplicity). If two vectors collide on all b entries in one group, the first is subtracted from the second, producing a vector with at least b all zero entries. These vectors are then again combined to produce more all zero entries and so forth until all a groups are eliminated to zero. However, as we add up vectors the noise increases. Overall, after ℓ addition levels the noise has standard deviation $\sqrt{2^\ell}\alpha q$. Our algorithm, too, will be parametrized by a positive integer $b \leq n$ (the window width), and $a := \lceil n/b \rceil$ (the addition depth).

Recall that the complexity of BKW algorithm is essentially q^b. However, b only depends on the ratio $\alpha q/\sqrt{2\pi}q = \alpha\sqrt{2\pi}$ and thus not on q. Hence, it is clear that applying modulus reduction before running the BKW algorithm may greatly improve its running time: b is preserved whilst q is reduced to p. However, instead of applying modulus reduction in 'one shot' prior to executing BKW, we propose switching to a lower precision only when needed. For this, we actually never switch the modulus but simply consider elements in $\mathbb{Z}_q$ 'through the perspective' of $\mathbb{Z}_p$. We then essentially only consider the top-most $\log_2 p$ bits of $\mathbb{Z}_q$.

Under this perspective, given samples of the form $(\mathbf{a}, c)$ we aim to produce $\left(\tilde{\mathbf{a}}, \tilde{c} = \langle\tilde{\mathbf{a}}, \mathbf{s}\rangle + \tilde{e}\right)$, where $\tilde{\mathbf{a}}$ is short enough, i.e.

$$|\langle\tilde{\mathbf{a}}, \mathbf{s}\rangle| \approx \sqrt{2^a}\alpha q. \tag{2}$$

Although other choices are possible, this choice means balancing the noise $\tilde{e}$ after a levels of addition and the contribution of $|\langle\tilde{\mathbf{a}}, \mathbf{s}\rangle|$ such that neither dominates. We call the term $\langle\tilde{\mathbf{a}}, \mathbf{s}\rangle$ the *rounding error*. So, condition (2) is such that after a levels of additions performed by the BKW algorithm the escalated initial noise and the noise coming from rounding errors have the same size.

2.2 Sample Reduction for Short Secrets

Let $(\mathbf{a}_0, c_0), \ldots, (\mathbf{a}_{m-1}, c_{m-1})$ be samples which follow $L_{\mathbf{s},\chi}$ or $\mathcal{U}(\mathbb{Z}_q^n) \times \mathcal{U}(\mathbb{Z}_q)$. We now explain how to produce samples $(\tilde{\mathbf{a}}_i, \tilde{c}_i)_{i\geq 0}$ that satisfy condition (2). For simplicity, we assume from now on that $p = 2^\kappa$. [2]

The main idea of the algorithm is to search for collisions among the first b components of samples $(\mathbf{a}_i, c_i)$ by only considering their top $\log_2 p$ bits. If such a collision is found, we proceed as in the normal BKW algorithm, i.e. we subtract the colliding samples to clear the first b components. In our case, we clear the top-most $\log_2 p$ bits of the first b components. Hence, instead of managing elimination tables for every bit of all components, we only manage elimination tables for the most significant κ bits. Put differently, all arithmetic is performed in $\mathbb{Z}_q$ but collisions are searched for in $\mathbb{Z}_p$ after rescaling or modulus switching.

As in [3], we realise the first stage of the BKW algorithm as a (recursively constructed) series of oracles $B_{\mathbf{s},\chi}(b, \ell, p)$. In our case, we have $0 \leq \ell < a$, where $B_{\mathbf{s},\chi}(b, a-1, p)$ produces the final output and $B_{\mathbf{s},\chi}(b, -1, p)$ calls the LWE oracle. We will make use of a set of tables T^ℓ (maintained across oracle calls) to store (randomly-chosen) vectors that will be used to reduce samples arising from our oracles. However, compared to [3] our oracles $B_{\mathbf{s},\chi}(b, \ell, p)$ take an additional parameter p which specifies the precision which we consider. Hence, if $p = q$ then we recover the algorithm from [3] where we perform no modulus reduction at all. In particular, $B_{\mathbf{s},\chi}(b, \ell, p)$ proceeds as follows:

1. For $\ell = -1$, we can obtain samples from $B_{\mathbf{s},\chi}(b, -1, p)$ by simply calling the LWE oracle $L_{\mathbf{s},\chi}$ and returning the output.
2. For $\ell = 0$, we repeatedly query the oracle $B_{\mathbf{s},\chi}(b, 0, p)$ to obtain (at most) $(p^b - 1)/2$ samples $(\mathbf{a}, c)$ with distinct non-zero vectors $\lfloor p/q \cdot \mathbf{a}_{(0,b)} \rceil$. We use these samples to populate the table T^0, indexed by $\lfloor p/q \cdot \mathbf{a}_{(0,b)} \rceil$. We store $(\mathbf{a}, c)$ in the table. During this course of this population, whenever we obtain a sample $(\mathbf{a}', c')$ from $B_{\mathbf{s},\chi}(b, -1, p)$, if $\left\lfloor p/q \cdot \mathbf{a}'_{(0,b)} \right\rceil$ (resp. the negation) match $\lfloor p/q \cdot \mathbf{a}_{(0,b)} \rceil$ 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 $B_{\mathbf{s},\chi}(b, 0, p)$. Note that, if $\lfloor p/q \cdot \mathbf{a}_{(0,b)} \rceil$ is zero, we return $(\mathbf{a}', c')$ as a sample from $B_{\mathbf{s},\chi}(b, 0, p)$. Further calls to the oracle $B_{\mathbf{s},\chi}(b, 0, p)$ proceed in a similar manner, but using (and potentially adding entries to) the same table T^0.

[2] While we do not have to restrict our attention to p of the form 2^κ, we choose it for ease of exposition and implementation.

3. For $0 < \ell < a$, we proceed as above: we make use of the table T^{ℓ} (constructed by calling $B_{\mathbf{s},\chi}(b, \ell-1, p)$ up to $(p^b-1)/2$ times) to reduce any output sample from $B_{\mathbf{s},\chi}(b, \ell-1, p)$ with $\lfloor p/q \cdot \mathbf{a}_{(b\cdot\ell, b\cdot\ell+b)} \rceil$ by an element with a matching such vector, to generate a sample returned by $B_{\mathbf{s},\chi}(b, \ell, p)$.

Pseudo-code for the modified oracle $B_{\mathbf{s},\chi}(b, \ell, p)$, for $0 \leq \ell < a$, is given in Algorithm 1.

Input: b – an integer $0 < b \leq n$
Input: ℓ – an integer $0 \leq \ell < a$
Input: p – an integer $0 < p \leq q$
begin
  $T^{\ell} \leftarrow$ table with p^b rows maintained across all runs of $B_{\mathbf{s},\chi}(b, \ell, p)$;
  repeat
    query $B_{\mathbf{s},\chi}(b, \ell-1, p)$ to obtain $(\mathbf{a}, c)$;
    $\mathbf{z} \leftarrow \left\lfloor \frac{p \cdot \mathbf{a}_{(b\cdot\ell, b\cdot(\ell+1))}}{q} \right\rceil$;
    if $\mathbf{z}$ *is all zero* **then**
      return $(\mathbf{a}, c)$;
    else if $T_{\mathbf{z}} \neq \varnothing$ **then**
      break;
    $T_{\mathbf{z}} \leftarrow (\mathbf{a}, c)$;
    $\overline{\mathbf{z}} \leftarrow \left\lfloor \frac{-p \cdot \mathbf{a}_{(b\cdot\ell, b\cdot(\ell+1))}}{q} \right\rceil$;
    $T_{\overline{\mathbf{z}}} \leftarrow (-\mathbf{a}, -c)$;
  until *the world ends*;
  $(\mathbf{a}', c') \leftarrow T_{\mathbf{z}}$;
  return $(\mathbf{a} - \mathbf{a}', c - c')$;

Algorithm 1: $B_{\mathbf{s},\chi}(b, \ell, p)$ for $0 \leq \ell < a$.

2.3 Picking p

Yet, we still have to establish the size of p to satisfy Condition 2. We note that in our approach we do not actually multiply by p/q. Let σ_r be the standard deviation of uniformly random elements in $\mathbb{Z}_{\lfloor q/p \rceil}$. Performing one-shot modulus switching in this setting would mean splitting $\mathbf{a}$ into two vectors, $\mathbf{a}'$ with the 'high order' bits and $\mathbf{a}''$ with 'low order' bits. The components of the latter would contribute to the final noise as the rounding error, the components of the former would be eliminated by BKW. The standard deviation of the components of $\mathbf{a}''$ is σ_r. For each component of $\mathbf{a}_{(i)}$ one-shot modulus switching would add a noise with standard deviation $\sigma_r\sigma_s$. Hence, after applying BKW to these pre-processed samples, the standard deviation of the noise contributed by modulus-switching in the final output would be

$$\sqrt{n \cdot 2^a \cdot \sigma_r^2\sigma_s^2} = \sqrt{a\, b \cdot 2^a \cdot \sigma_r^2\sigma_s^2}. \tag{3}$$

However, as the following lemma establishes, we may consider smaller p because the final noise contributed by modulus switching under Algorithm 1 is smaller than in (3). This is because if $(\tilde{\mathbf{a}}_i, \tilde{c}_i)$ are final output samples then the entries $\tilde{\mathbf{a}}_{i,(b\cdot a-1)}$ will be significantly smaller than $\tilde{\mathbf{a}}_{i,(0)}$.

Yet, to formalise this, we need to make a (standard) simplifying assumption, namely that the outputs of the BKW algorithm (at every stage) are independent. That is, we make the assumption that, during the course of the algorithm described, all components of each sample from $B_{\mathbf{s},\chi}(b, \ell, p)$ are independent from every other sample. We emphasize that similar assumptions are standard in treatments of combinatorial algorithms for LPN/LWE (cf. [3,11]).

Assumption 1 *We assume that all outputs of $B_{\mathbf{s},\chi}(b,\ell,p)$ are independent.*

Remark 1. Calculations show that if we consider any two of the outputs of our algorithm, the expected proportion of shared elements is very small (a more detailed investigation of such effects is currently in preparation as part of an independent work). In the event of two final samples sharing one or a small number of error elements, the combined noise of these two samples remains weakly dependent. Indeed, in [11], the authors presented and implemented a heuristic algorithm related to BKW in which all attempts at preserving independence between samples were abandoned, yet they report that the results were indistinguishable from the independent or weakly-dependent BKW samples. In the course of extensive experimentation, no deviation from the behaviour expected from the presumed independence of samples was observed.

Assumption 1 allows to establish the following lemma:

Lemma 1. *Let $n \geq 1$ be the dimension of the* LWE *secret vector, q be a modulus, $b \in \mathbb{Z}$ with $1 \leq b \leq n$. Let also σ_r be the standard deviation of uniformly random elements in $\mathbb{Z}_{\lfloor q/p \rceil}$. Under Assumption 1, the components of $\tilde{\mathbf{a}} = \mathbf{a} - \mathbf{a}'$ returned by $B_{\mathbf{s},\chi}(b,\ell,p)$ satisfy:*

$$\mathrm{Var}(\tilde{\mathbf{a}}_{(i)}) = 2^{\ell - \lfloor i/b \rfloor}\sigma_r^2, \text{ for } 0 \leq \lfloor i/b \rfloor \leq \ell$$

and $\mathrm{Var}\big(\mathcal{U}(\mathbb{Z}_q)\big)$ *for* $\lfloor i/b \rfloor > \ell$.

Proof. We assume $b = 1$ without loss of generality and proceed by induction on ℓ.

Initialization. If $\ell = 0$, then $i = 0$. The output of $B_{\mathbf{s},\chi}(b,0,p)$ is the sum of two random vectors in $\mathbb{Z}_q^n$ which collide in component zero when considered in $\mathbb{Z}_p$. The variance of the result is hence that of a random element in $\mathbb{Z}_{\lfloor q/p \rceil}$, i.e. σ_r^2, in component zero, all other components follow the uniform distribution in $\mathbb{Z}_q$.

If $\ell = 1$, then $i = 0$ and 1. Also, we have two elimination tables T^0 and T^1. Outputs of $B_{\mathbf{s},\chi}(b,1,p)$ are the sum of two outputs of $B_{\mathbf{s},\chi}(b,0,p)$. Under Assumption 1 these are independent and the sum of their variances is the variance of their sum. The variance of $\tilde{\mathbf{a}}_{(0)}$ is hence $2\sigma_r^2$ and $\tilde{\mathbf{a}}_{(1)}$ has variance σ_r^2 similarly to case $\ell = 0$. All other components are uniformly random in $\mathbb{Z}_q$.

Induction. More generally, for $\ell > 0$ the output of $B_{\mathbf{s},\chi}(b,\ell,p)$ is the sum of two outputs of $B_{\mathbf{s},\chi}(b,\ell-1,p)$. Hence, its components satisfy $\mathrm{Var}(\tilde{\mathbf{a}}_{(i)}) = 2 \cdot 2^{\ell-1-i}\sigma_r^2$ for $0 < i < \ell$ and σ_r^2 for $\mathbf{a}_{(\ell)}$. □

Using Lemma 1 we may adapt our choice of p, because the noise contributed by modulus switching for a given p is smaller:

Corollary 1. *Let $n \geq 1$ be the dimension of the* LWE *secret vector, q be a modulus, $b \in \mathbb{Z}$ with $1 \leq b \leq n$. Let σ_r be the standard deviation of uniformly random elements in $\mathbb{Z}\lfloor q/p \rceil$ and σ_s be the standard deviation of the distribution from which the secret $\mathbf{s}$ is sampled. Let $(\tilde{a},\tilde{c})$ be an output of $B_{\mathbf{s},\chi}(b,a-1,p)$. Under Assumption 1, the noise added by lazy modulus switching in the final output of $B_{\mathbf{s},\chi}(b,a-1,p)$, that is $|\langle \tilde{\mathbf{a}}, \mathbf{s} \rangle|$, has standard deviation*

$$\sqrt{b \cdot \left(\sum_{i=0}^{a-1} 2^{a-i-1} \right) \cdot \sigma_r^2\sigma_s^2} = \sqrt{b \cdot (2^a - 1) \cdot \sigma_r^2\sigma_s^2}.$$

Proof. From Lemma 1, we are adding n (assumed to be) independent random variables, each of which takes the form $\tilde{\mathbf{a}}_i \cdot \mathbf{s}_i$ where $\tilde{\mathbf{a}}_i$ is distributed according to the interval of b elements in which i lies. The corollary then follows by adding the variances of b such random variables from each interval. □

Now, compare Corollary 1 with the standard deviation in (3). We see that the standard deviation obtained using our lazy modulus switching is divided by a factor $\sqrt{a}$ w.r.t. to a naive use of modulus-switching, i.e. as in (3). As a consequence, we may reduce p by a factor $\sqrt{a}$.

3 Improved Algorithm: Stunting Growth by Unnatural Selection

Based on the strategy in the previous section, we now introduce a pre-processing step which allows us to further reduce the magnitude of the noise present in the outputs of $B_{\mathbf{s},\chi}(b, a-1, p)$ by reducing rounding errors further. For this, it will be useful to establish notation to refer to various components of $\mathbf{a}_i$ in relation to $B_{\mathbf{s},\chi}(b, \ell, p)$.

Children: are all those components with index $j < b \cdot \ell$, i.e. those components that were reduced by some $B_{\mathbf{s},\chi}(b, k, p)$ with $k < \ell$: *they grow up so quickly.*

Parents: are those components of $\mathbf{a}_i$ with index $b \cdot \ell \leq j < b \cdot \ell + b$, i.e. those components among which collisions are searched for in $B_{\mathbf{s},\chi}(b, \ell, p)$: *collisions among parents produce children.*

Strangers: with respect to $B_{\mathbf{s},\chi}(b, \ell, p)$ are all other components $j \geq b \cdot \ell + b$: *they are indifferent towards each other.*

So, for example, if $n = 10$ and $b = 2$ and we are considering $\ell = 2$ then $\mathbf{a}_{(0-3)}$ are child components, $\mathbf{a}_{(4-5)}$ are parents and $\mathbf{a}_{(6-9)}$ are strangers (cf. Figure 1).

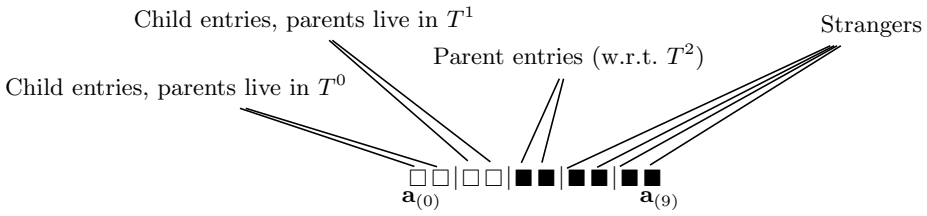

Fig. 1. Children, parents and strangers.

3.1 The Basic Idea

For the general idea and intuition, assume $b = 1$ and that $\tilde{\mathbf{a}}_i$ are outputs of $B_{\mathbf{s},\chi}(b, 0, p)$ and we hence have $\mathrm{Var}(\tilde{\mathbf{a}}_{i,(0)}) = \sigma_r^2$. Now, some of these $\tilde{\mathbf{a}}_i$ will be stored in Table T^1 by $B_{\mathbf{s},\chi}(b, 1, p)$ based on the value in the parent component $\tilde{\mathbf{a}}_{i,(1)}$. All future outputs of $B_{\mathbf{s},\chi}(b, 1, p)$ which collide with $\tilde{\mathbf{a}}_i$ in the parent component at index 1 will have $\tilde{\mathbf{a}}_i$ added/subtracted to it, we are hence adding a value with $\mathrm{Var}(\tilde{\mathbf{a}}_{i,(0)}) = \sigma_r^2$ in index 0.

Now, however, if the $\tilde{\mathbf{a}}_{i,(0)}$ happened to be unusually short, all $B_{\mathbf{s},\chi}(b, \ell, p)$ for $\ell > 0$ would output vectors with a shorter $\tilde{\mathbf{a}}_{i,(0)}$ added/subtracted in, i.e. would also have unusually small child components (although to a lesser degree). That is, improving the outputs of $B_{\mathbf{s},\chi}(b, 1, p)$ – i.e. decreasing the magnitude of the $\tilde{\mathbf{a}}_{i,(0)}$ stored in T^1 – has a knock-on effect on all later outputs. More generally, improving the outputs of $B_{\mathbf{s},\chi}(b, \ell, p)$ will improve the outputs of $B_{\mathbf{s},\chi}(b, k, p)$ for $k > \ell$.

On the other hand, improving the outputs of $B_{\mathbf{s},\chi}(b, \ell, p)$ where ℓ is small, is easier than for larger values of ℓ. In the algorithm as described so far, when we obtain a collision between a member

of T^ℓ and an output $(\mathbf{a}_i, c_i)$ of $B_{\mathbf{s},\chi}(b, \ell-1, p)$, we reduce $(\mathbf{a}_i, c_i)$ using the colliding member of T^ℓ, retaining this member in the table. Alternatively we can reduce $(\mathbf{a}_i, c_i)$ using the *in-situ* table entry, replace the table entry with (the now reduced) $(\mathbf{a}_i, c_i)$ and return the former table entry as the output of $B_{\mathbf{s},\chi}(b, \ell, p)$. If we selectively employ this alternative strategy using the relative magnitudes of the child components of $(\mathbf{a}_i, c_i)$ and the table entry as a criterion, we can improve the 'quality' of our tables as part of a pre-processing phase.

That is, in $B_{\mathbf{s},\chi}(b, \ell, p)$ for each collision in a parent component we may inspect the child components for their size and keep that in T^ℓ where the child components are smallest. Phrased in the language of 'children' and 'parents': we do not let 'nature', i.e. randomness, run its course but intervene and select children based on their size. As the number of child components is $b \cdot \ell$ it becomes more difficult as ℓ increases to find vectors where all child components are short.

3.2 Algorithms

This leads to a modified algorithm $B_{small,\mathbf{s},\chi}(b, \ell, p)$ given in Algorithm 2. Using Algorithms 1 and 2 we may then present our revised version of the BKW algorithm in Algorithm 3 where we first use Algorithm 2 to produce 'good' tables and then use Algorithm 1 to sample $(\tilde{\mathbf{a}}_i, \tilde{c}_i)$ as before.

Input: b – an integer $0 < b \leq n$
Input: ℓ – an integer $0 \leq \ell < a$
Input: p – an integer $0 < p \leq q$
begin
  $T^\ell \leftarrow$ table with p^b rows maintained across all runs of $B_{small,\mathbf{s},\chi}(b, \ell, p)$;
  Find $(\mathbf{a}', c') \leftarrow T^\ell_{\mathbf{z}}$ that collides with a fresh sample $(\mathbf{a}, c)$ from $B_{\mathbf{s},\chi}(b, \ell-1, p)$ as in Algorithm 1;
  if $\sum_{i=0}^{b\cdot\ell-1} \left|\mathbf{a}'_{(i)}\right| > \sum_{i=0}^{b\cdot\ell-1} \left|\mathbf{a}_{(i)}\right|$ **then**
    $T^\ell_{\mathbf{z}} \leftarrow (\mathbf{a}, c)$;
  return $(\mathbf{a} - \mathbf{a}', c - c')$;

Algorithm 2: $B_{small,\mathbf{s},\chi}(b, \ell, p)$ for $0 \leq \ell < a$.

Input: b – an integer $0 < b \leq n$
Input: a – an integer such that $ab = n$
Input: p – an integer $0 < p < q$
Input: o – an integer $0 \leq o$
Input: m – an integer $0 \leq m$
begin
  $o_t \leftarrow o/(a+1)$;
  `// populate elimination tables with random entries`
  for $0 \leq i < o_t$ **do**
    $(\tilde{\mathbf{a}}, c) \leftarrow B_{\mathbf{s},\chi}(b, a-1, p)$; `//` $(\tilde{\mathbf{a}}, c)$ `is discarded`
  `// sample small entries`
  for $0 \leq i < a$ **do**
    for $0 \leq j < o_t$ **do**
      $(\tilde{\mathbf{a}}, c) \leftarrow B_{small,\mathbf{s},\chi}(b, i, p)$; `//` $(\tilde{\mathbf{a}}, c)$ `is discarded`
  for $0 \leq i < m$ **do**
    $(\tilde{\mathbf{a}}_i, c_i) \leftarrow B_{\mathbf{s},\chi}(b, a-1, p)$;
  Run distinguisher on c_i and return output;

Algorithm 3: BKW with lazy modulus switching.

3.3 Picking p

It remains to be established what the effect of such a strategy is, i.e. how fast children grow up or how fast rounding errors accumulate. In particular, given n vectors $\mathbf{x}_i$ sampled from some distribution $\mathcal{D}$ where each component has standard deviation σ, i.e. $\mathrm{Var}(\mathbf{x}_{i,(j)}) = \sigma^2$ we are interested in the standard deviation σ_n of each component for $\mathbf{x}^* = \min_{abs}(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1})$ where $\min_{abs}$ picks that vector where $\sum_{j=0}^{b\cdot\ell-1} \left|\mathbf{x}_{(j)}\right|$ is minimal. At this point we know no closed algebraic expression for σ_n. However, we found – as detailed below – that σ_n can be estimated as follows:

Assumption 2 *Let the vectors $\mathbf{x}_0, \ldots, \mathbf{x}_{n-1} \in \mathbb{Z}_q^\tau$ be sampled from some distribution $\mathcal{D}$ such that $\sigma^2 = \mathrm{Var}(\mathbf{x}_{i,(j)})$ where $\mathcal{D}$ is any distribution on (sub-)vectors observable in Algorithm 3. Let $\mathbf{x}^* = \min_{abs}(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1})$ where $\min_{abs}$ picks that vector $\mathbf{x}^*$ with $\sum_{j=0}^{b\cdot\ell-1} \left|\mathbf{x}^*_{(j)}\right|$ minimal. The stddev $\sigma_n = \sqrt{\mathrm{Var}(\mathbf{x}^*_{(0)})} = \cdots = \sqrt{\mathrm{Var}(\mathbf{x}^*_{(\tau-1)})}$ of components in $\mathbf{x}^*$ satisfies*

$$\sigma/\sigma_n \geq c_\tau \sqrt[\tau]{n} + (1 - c_\tau)$$

with c_τ as in Table 1 for $\tau \leq 10$ and

$$c_\tau = 0.20151418166952917\sqrt{\tau} + 0.32362108131969386 \approx \frac{1}{5}\sqrt{\tau} + \frac{1}{3}$$

otherwise.

τ	1	2	3	4	5
c_τ	0.405799353869	0.692447899282	0.789885269135	0.844195936036	0.854967912468
τ	6	7	8	9	10
c_τ	0.895446987232	0.91570933651	0.956763578012	0.943424544282	0.998715322134

Table 1. c_τ for small values of τ

We arrive at the approximation in Assumption 2 as follows. We chose $\mathcal{D} = \mathcal{U}(\mathbb{Z}_{2^{32}})$ and observed that σ/σ_n behaves linearly when $\tau = 1$ (cf. Figure 2). As we increase τ we expect the problem of finding short vectors to become exponentially harder. We also require $\sigma/\sigma_1 = 1$ for any τ. The approximation $c_\tau \sqrt[\tau]{x} + (1 - c_\tau) = \sigma/\sigma_n$ satisfies all these conditions. Furthermore, experimental evidence suggests that there exist c_τ such that a function of this form approximates the observed data closely (cf. Figure 2). We then used curve fitting (as implemented in Sage [23] via calls to SciPy [15] which in turn calls MINPACK [20]) on experimental data for $1 \leq n \leq 128$ to find c_τ for $1 \leq \tau \leq 512$; Table 1 lists the first 10 such values. The expression $c_\tau \approx 1/5\sqrt{\tau} + 1/3$ was recovered by curve fitting the pairs $(1, c_1), \ldots, (512, c_{512})$ recovered before (cf. Figure 3 for a plot of this data and $1/5\sqrt{\tau} + 1/3$). However, for small values of τ this expression diverges too much from the observed, which is why we are using explicit values instead. Finally, we assume that all distributions observed during the course of running our algorithm 'shrink' at least as fast as the uniform distribution. An assumption we experimentally verified for the normal distribution and which seems to be confirmed by our experimental results in Section 5.

With Assumption 2 we can now estimate the size of the entries of the variance matrix associated with our elimination tables. That is, a matrix $\mathbf{M}$ where the entry $\mathbf{M}_{(i,j)}$ holds the variance of entries $(b \cdot j, \ldots, b \cdot j + b - 1)$ in T^i.

It is clear that $\mathbf{M}_{(i,j)} = \mathrm{Var}(\mathcal{U}(\mathbb{Z}_q))$ for $0 \leq i < a$ and $i \leq j$ as no reductions take place for entries on and above 'the main diagonal'. Now, in Algorithm 3 the child components in T^1 are reduced by calling Algorithm 2 $o/(a+1)$ times. Each table T^ℓ has $p^b/2$ rows and we can hence

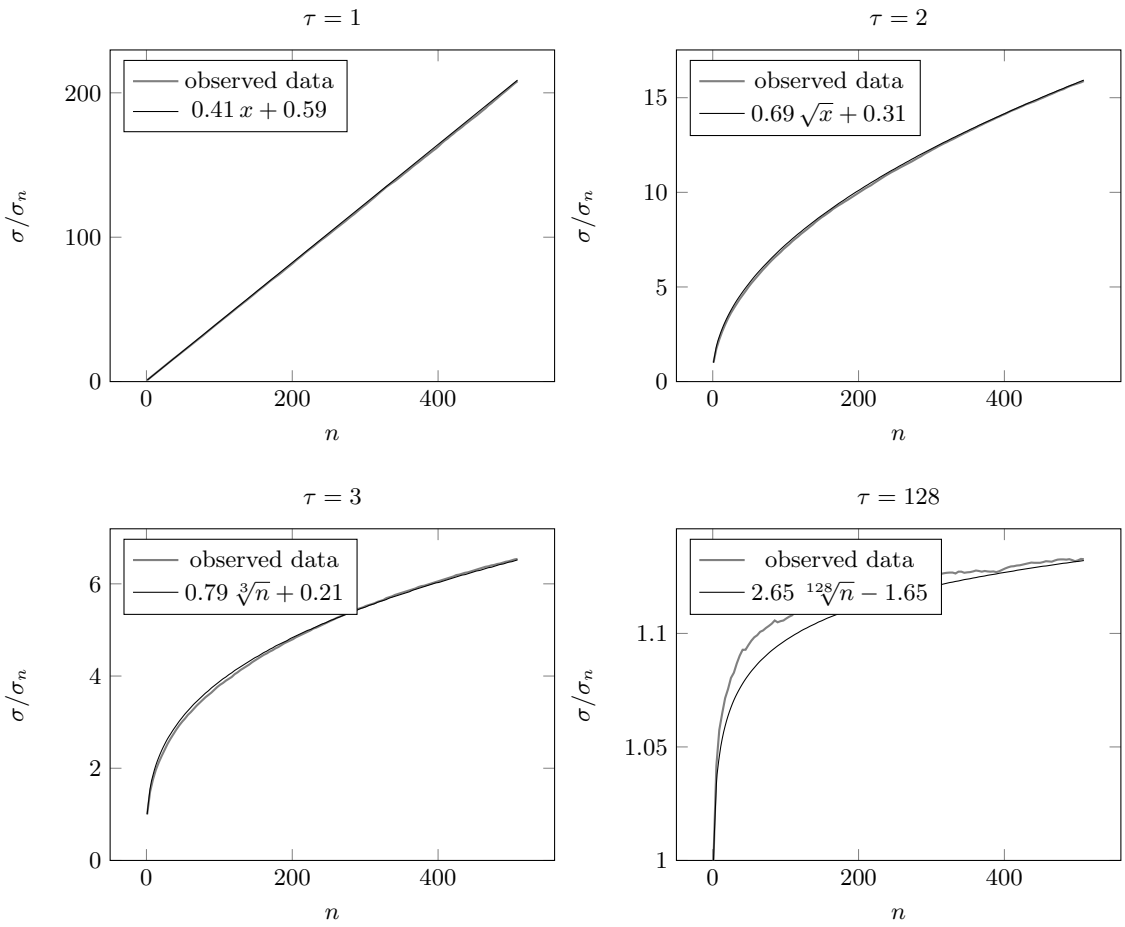

Fig. 2. σ/σ_n for $1 \leq \tau \leq 3$ and $\tau = 128$.

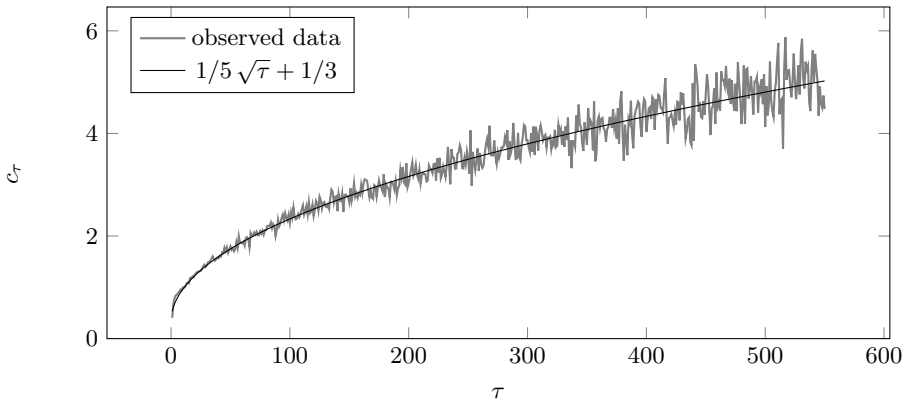

Fig. 3. c_τ in function of τ

apply Assumption 2 on T^1 with $\tau = b$ and $n = \frac{2o}{(a+1)p^b} + 1$ uniform samples (i.e. $\mathcal{D}$ is the uniform distribution) with standard deviation σ_r. Note that this assumes (idealistically) that each table entry is 'hit' exactly n times during this process. While the expected value of 'hits' each table entry receives is n, ideally we wish to ensure that the majority of table entries are 'hit' at least a certain number of times. Clearly, the number of 'hits' for a given table entry is governed by a binomial distribution - if we consider the problem from a 'balls and bins' perspective, we have the random and independent placing of $o/(a+1)$ balls into $p^b/2$ bins. Then we can approximate the expected number of bins containing j balls by a Poisson random variable with parameter $o/((a+1) \cdot p^b/2)$, implying we can approximate the number of such bins by

$$\frac{(o/((a+1) \cdot p^b/2))^j}{j!} \cdot e^{-\frac{2o}{(a+1) \cdot p^b}}$$

Thus we can approximate the number of bins containing less than M balls by

$$p^b/2 \cdot e^{-\frac{o}{(a+1) \cdot p^b/2}} \cdot \sum_{k=0}^{M-1} \frac{((o/((a+1) \cdot p^b/2)))^k}{k!}$$

Now, it is well known that when the parameter is large enough, the Poisson distribution itself may be approximated closely by a Gaussian distribution. The parameter for the Poisson distribution is $\frac{o}{(a+1) \cdot p^b/2}$ hence, by a standard result, we can approximate this Poisson distribution by a Gaussian of mean $\frac{o}{(a+1) \cdot p^b/2}$ and of variance also $\frac{o}{(a+1) \cdot p^b/2}$.

Thus, under the assumption that these approximations hold, we can approximate the probability that a given bin contains less than x balls by the standard CDF for Gaussians:

$$\frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{x - \frac{o}{(a+1) \cdot p^b/2}}{\sqrt{\frac{2 \cdot o}{(a+1) \cdot p^b/2}}}\right)\right).$$

However, in Algorithm 4 below we use the mean and take the distribution of balls in bins into account by reducing the number of observed samples by a fudge factor of 2 in our calculations.

By Assumption 2 we hence get

$$\mathbf{M}_{(1,0)} = \frac{\sigma_r^2}{\left(c_b \sqrt[b]{n} + 1 - c_b\right)^2}$$

Moving on to $\mathbf{M}_{(2,0)}$ we have $\tau = 2b$. Hence, using the same reasoning as in Lemma 1 we expect

$$\mathbf{M}_{(2,0)} = \frac{\sigma_r^2 + \mathbf{M}_{(1,0)}}{\left(c_{2b} \sqrt[2b]{n} + 1 - c_{2b}\right)^2} = \frac{\sigma_r^2 + \frac{\sigma_r^2}{\left(c_b \sqrt[b]{n} + 1 - c_b\right)^2}}{\left(c_{2b} \sqrt[2b]{n} + 1 - c_{2b}\right)^2}$$

and so on for $\mathbf{M}_{(3,0)}, \mathbf{M}_{(4,0)}, \ldots$.

An algorithm for constructing $\mathbf{M}$ is given in Algorithm 4 which we expect this algorithm to give a reasonable approximation of the variances of components of entries in T^ℓ and back up this expectation with empirical evidence in Section 5.

Using the matrix $\mathbf{M}$ computed by Algorithm 4, we can estimate the variances of components of $\tilde{\mathbf{a}}_i$ as output by $B_{\mathbf{s},\chi}(b, a-1, p)$. This result follows immediately from Assumption 2.

Lemma 2. *Let $n \geq 1, q$ be a modulus, $b \in \mathbb{Z}$ with $1 \leq b \leq n$ and σ_r be the standard deviation of $\mathcal{U}(\mathbb{Z}_{\lfloor q/p \rceil})$. Define $a := \lceil n/b \rceil$ and pick some $p < q$ and let $\mathbf{M}$ be the output of Algorithm 4*

begin
  $T \leftarrow 2 \cdot p^b/2$; `// fudge factor: 2`
  $n \leftarrow \frac{m^*}{(a+1)\cdot T} + 1$;
  $\mathrm{Var}_{red} = \mathrm{Var}(\mathcal{U}(\mathbb{Z}_{\lfloor q/p \rceil})) = \sigma_r^2$; `// the var. of fresh red. elements`
  $\mathbf{M}$ is an $a \times a$ matrix;
  for $0 \leq r < a$ **do**
    for $0 \leq c < a$ **do**
      $\mathbf{M}_{(r,c)} \leftarrow \mathrm{Var}(\mathcal{U}(\mathbb{Z}_q))$; `// el. on and above main diag. not red.`
  for $1 \leq t < a$ **do**
    `// row` t `= sum of prev. rows + 1 fresh el. for each index`
    for $0 \leq i < t$ **do**
      $\mathbf{M}_{(t,i)} \leftarrow \mathrm{Var}_{red} + \sum_{j=i+1}^{t-1} \mathbf{M}_{(j,i)}$;
    $\tau \leftarrow b \cdot \ell$;
    for $0 \leq i < t$ **do**
      $\mathbf{M}_{(t,i)} \leftarrow \frac{\mathbf{M}_{(t,i)}}{\left(c_\tau \sqrt[\ell]{n}+1-c_\tau\right)^2}$;

Algorithm 4: Constructing $\mathbf{M}$.

under these parameters. Let $(\tilde{\mathbf{a}}_i, c_i)$ *be samples returned by* $B_{\mathbf{s},\chi}(b, a-1, p)$*. Finally, define* $\mathbf{v}$ *as the* $a-$*vector of variances of the components of* $\tilde{\mathbf{a}}$ *where* $\mathbf{v}_{(k)}$ *holds the variance of the components* $\tilde{\mathbf{a}}_{(b\cdot k)}$ *to* $\tilde{\mathbf{a}}_{(b\cdot k+b-1)}$*. Under Assumption 2, the components of* $\mathbf{v}$ *satisfy:*

$$\mathbf{v}_{(i)} = \sigma_r^2 + \sum_{j=i+1}^{a} \mathbf{M}_{(j,i)}.$$

This now allows us to given an expression for the noise distribution output by $B_{\mathbf{s},\chi}(b, a-1, p)$.

Lemma 3. *Let* $n \geq 1$ *be the dimension of the* LWE *secret vector,* q *be a modulus,* $b \in \mathbb{Z}$ *with* $1 \leq b \leq n$*. Define* $a := \lceil n/b \rceil$ *and pick some* $p < q$ *and let* $\mathbf{v}$ *be as in Lemma 2. Let* $(\tilde{\mathbf{a}}_i, \tilde{c}_i)$ *be outputs of* $B_{\mathbf{s},\chi}(b, a-1, p)$*. We assume that Assumptions 1 and 2 hold. Then as* a *increases the distribution of* $\tilde{c}_i$ *approaches a discrete Gaussian distribution modulo* q *with standard deviation*

$$\sigma_{total} := \sqrt{2^a\sigma + b\,\sigma_r^2\sigma_s^2 \sum_{i=0}^{a-1} \mathbf{v}_{(i)}} \leq \sqrt{2^a\sigma + (2^a-1)\cdot b \cdot \sigma_r^2\sigma_s^2}.$$

Proof. The standard deviation follows from Assumption 1 and Lemma 2. Since the distribution is formed by adding up 2^a vectors it approaches a discrete Gaussian distribution when considered over $\mathbb{Z}$ as a increases by the Central Limit Theorem. □

Assumption 3 *We assume that Lemma 3 holds for* $128 \leq n$*, i.e. the values of* n *considered in this work.*

4 Complexity

Finally, we analyse the complexity of the presented algorithms. To do so, we assume that Assumptions 1, 2, and 3 hold. Lemma 3 allows us to estimate the numbers of samples needed to distinguish the outputs of $B_{\mathbf{s},\chi}(b, a-1, p)$ if $B_{\mathbf{s},\chi}(b, -1, p)$ returns LWE samples from uniform. For this, we

rely on standard estimates [17] for the number of samples required to distinguish. This estimate provides a good approximation for the advantage obtainable in distinguishing between $\mathcal{U}(\mathbb{Z}_q)$ and a discrete Gaussian reduced mod q with standard deviation σ_{total}. In particular, we compute the advantage as

$$\text{Adv} = \exp\left(-\pi\left(\frac{\sigma_{total}\cdot\sqrt{2\pi}}{q}\right)^2\right).$$

We can now state the overall complexity of running the algorithm in Theorem 1. Remark that the proof of next two results are omitted; they follow by an easy adaptation of the proof of Lemma 2 in [3].

Theorem 1. *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 $\mathbb{Z}_{\lfloor q/p_{\text{small}}\rceil}$. Define $a := \lceil n/b \rceil$ and pick a pair (p_{small}, m^*) such that $b\,\sigma_r^2\sigma_s^2\sum_{i=0}^{a-1}\mathbf{v}_{(i)} \leq 2^a\sigma$, where $\mathbf{v}_{(i)}$ is defined as in Lemma 3. 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 $\leq \sqrt{2^{a+1}}\cdot\sigma$. Furthermore, this costs*

$$\frac{p_{\text{small}}^b}{2}\cdot\left(\frac{a(a-1)}{2}\cdot(n+1)\right) + (m+m^*)\;n\,a$$

additions in $\mathbb{Z}_q$ and $a\cdot\left(\frac{p_{\text{small}}^b}{2}\right) + m + m^*$ calls to $L_{\mathbf{s},\chi}$.

The memory requirement for storing each table is established in Corollary 2 below.

Corollary 2. *The memory required to store the table T^i is upper-bounded by*

$$\frac{p_{\text{small}}^b}{2}\cdot a\cdot(n+1)$$

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

To clarify the impact of Theorem 1, we consider $m^* = 0$ – i.e. the case discussed in Section 2 – on classical parameters of LWE.

Corollary 3. *Let $q \approx n^c$, for some constant $c > 0$, and $\alpha = n^{1/2-c}$ such that $\sigma \approx \alpha q \approx \sqrt{n}$. Furthermore, let $a = \log_2 n$ and $b = n/\log_2 n$ be the usual choices of parameters for BKW. Assume σ_s does not depend on n. Then, solving Decision-*LWE *costs at most*

$$\mathcal{O}\left(2^{n\left(c+\frac{\log_2 d-\frac{1}{2}\log_2\log_2 n}{\log_2 n}\right)}\cdot n\,\log_2^2 n\right)$$

operations in $\mathbb{Z}_q$. We also need to store $\mathcal{O}\left(2^{n\left(c+\frac{\log_2 d-\frac{1}{2}\log_2\log_2 n}{\log_2 n}\right)}\cdot n\,\log_2 n\right)$ elements in $\mathbb{Z}_q$.

Proof. First, we recall that the time complexity of the BKW algorithm, under these parameters and as analysed in [3, Corollary 3], is $\approx a^2\,n\,q^b$. Note that the memory needed is also $\approx a\,n\,q^b$. With the parameters considered, this yields a time complexity dominated by $\mathcal{O}\left(n^{cn/\log_2 n}\cdot n\,\log_2^2 n\right) = \mathcal{O}\left(2^{c\,n}\cdot n\,\log_2^2 n\right)$.

This can be improved by first performing a one-shot modulus switching, as explained in Section 2.3, and then using BKW on this pre-processed instances. A good choice is to take $p_{\text{small}} \approx$

$\min\{q,\ \frac{\sigma_s}{\sigma}\sqrt{\frac{n}{12}}\cdot q\}$, which simplifies to $\min\{q,\ \frac{\sigma_s}{\sqrt{12}}\cdot q\}$ or $d\cdot q$, with $0 < d \leq 1$, under these parameter choices. This allows to reduces the complexity to

$$\mathcal{O}\left((dn^c)^{n/\log_2 n}\cdot n\,\log_2^2 n\right) = \mathcal{O}\left(d^{n/\log_2 n}\cdot 2^{cn}\cdot n\,\log_2^2 n\right) = \mathcal{O}\left(2^{n\left(c+\frac{\log_2 d}{\log_2 n}\right)}\cdot n\,\log_2^2 n\right).$$

Since $\log_2 d < 0$, this indeed improves the complexity of the plain BKW.

Applying lazy modulus switching, once can reduce p_{small} by an additional factor of $\sqrt{a} = \sqrt{\log_2 n}$ (Corrolary 1). This gives:

$$\mathcal{O}\left(\left(\frac{dn^c}{\sqrt{\log_2 n}}\right)^{n/\log_2 n}\cdot n\,\log_2^2 n\right) = \mathcal{O}\left(2^{n\left(c+\frac{\log_2 d'}{\log_2 n}\right)}\cdot n\,\log_2^2 n\right),\ \text{with } d' = \left(\frac{d}{\sqrt{\log_2 n}}\right).$$

Finally $\log_2 d' = \log_2 d - \frac{1}{2}\log_2\log_2 n$, and then:

$$\mathcal{O}\left(\left(\frac{dn^c}{\sqrt{\log_2 n}}\right)^{n/\log_2 n}\cdot n\,\log_2^2 n\right) = \mathcal{O}\left(2^{n\left(c+\frac{\log_2 d-\frac{1}{2}\log_2\log_2 n}{\log_2 n}\right)}\cdot n\,\log_2^2 n\right).$$

The very same argument yields the memory complexity announced. □

5 Implementation

We implemented the algorithm presented in this work, i.e. Algorithm 3, to verify the assumptions made in this work and to confirm the expected behaviour of the algorithm. Our implementation supports machine size ring elements and integer values for b. We mention that our implementation is not optimised and hence we do not report CPU times. Our implementation is available at `http://bitbucket.org/malb/bkw-lwe`.

5.1 Independence Assumption

To verify the independence assumption, i.e. Assumption 1, and in particular that the noise part behaves like discrete Gaussian modulo q with $s = \sqrt{2^a}\alpha q$, we ran our implementation for $q = 4093, \sigma = 3.0$, and $b = 2$ with $n = 10$, 25 and 30. In each case, we asked for 2^{20} samples and extracted the noise by computing $c_i - \langle \tilde{\mathbf{a}}_i, \mathbf{s}\rangle$ and analysed the resulting distributions. In all our experiments, the noise followed a discrete Gaussian closely. In Figure 4 we plot a histogram for the experimental data (in gray) for the case $n = 30$ and the expected distribution for $\mathcal{N}(0, \sqrt{2^a}q)$.

5.2 Correctness of Algorithm 4

The behaviour our algorithm depends critically on the quality of approximation made in Algorithm 4. We hence verified that the matrix $\mathbf{M}$ returned by that algorithm matches the actual variances observed in practice.

We start, with an example where the prediction is quite precise. We considered the parameters $q = 65521$, $a = 20$, $b = 2$, $p = 2^{11}$ and $o = 2^{26}$. Algorithm 4 returns the following matrix

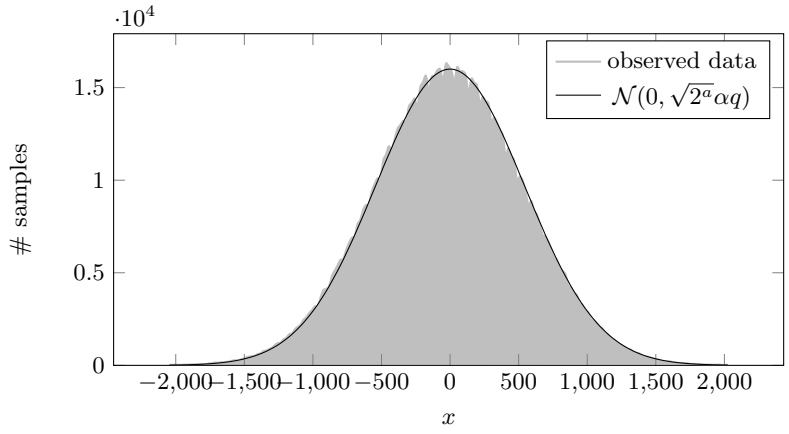

Fig. 4. Distribution of $c_i - \langle \tilde{\mathbf{a}}_i, \mathbf{s} \rangle$ for parameters $q = 4093, n = 30, a = 15, \sigma = 3.0$.

$$\log_2 \mathbf{M} = \begin{pmatrix}
28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
6.6 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
7.7 & 7.0 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
8.6 & 7.9 & 7.1 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
9.5 & 8.9 & 8.1 & 7.2 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
10.5 & 9.8 & 9.0 & 8.1 & 7.2 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
11.4 & 10.8 & 10.0 & 9.1 & 8.2 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
12.4 & 11.7 & 10.9 & 10.0 & 9.1 & 8.2 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
13.3 & 12.7 & 11.9 & 11.0 & 10.1 & 9.2 & 8.2 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
14.3 & 13.6 & 12.8 & 11.9 & 11.1 & 10.1 & 9.2 & 8.3 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
15.3 & 14.6 & 13.8 & 12.9 & 12.0 & 11.1 & 10.2 & 9.2 & 8.3 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
16.2 & 15.6 & 14.7 & 13.9 & 13.0 & 12.1 & 11.1 & 10.2 & 9.2 & 8.3 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
17.2 & 16.5 & 15.7 & 14.8 & 13.9 & 13.0 & 12.1 & 11.2 & 10.2 & 9.3 & 8.3 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
18.2 & 17.5 & 16.7 & 15.8 & 14.9 & 14.0 & 13.1 & 12.1 & 11.2 & 10.2 & 9.3 & 8.3 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
19.1 & 18.5 & 17.7 & 16.8 & 15.9 & 15.0 & 14.0 & 13.1 & 12.2 & 11.2 & 10.2 & 9.3 & 8.3 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
20.1 & 19.4 & 18.6 & 17.8 & 16.9 & 15.9 & 15.0 & 14.1 & 13.1 & 12.2 & 11.2 & 10.3 & 9.3 & 8.3 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
21.1 & 20.4 & 19.6 & 18.7 & 17.8 & 16.9 & 16.0 & 15.0 & 14.1 & 13.1 & 12.2 & 11.2 & 10.3 & 9.3 & 8.3 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
22.1 & 21.4 & 20.6 & 19.7 & 18.8 & 17.9 & 17.0 & 16.0 & 15.1 & 14.1 & 13.2 & 12.2 & 11.2 & 10.3 & 9.3 & 8.3 & 7.4 & 28.4 & 28.4 & 28.4 \\
23.0 & 22.4 & 21.6 & 20.7 & 19.8 & 18.9 & 17.9 & 17.0 & 16.1 & 15.1 & 14.1 & 13.2 & 12.2 & 11.3 & 10.3 & 9.3 & 8.3 & 7.4 & 28.4 & 28.4 \\
24.0 & 23.3 & 22.5 & 21.7 & 20.8 & 19.9 & 18.9 & 18.0 & 17.0 & 16.1 & 15.1 & 14.2 & 13.2 & 12.2 & 11.3 & 10.3 & 9.3 & 8.3 & 7.4 & 28.4
\end{pmatrix}$$

whereas our implementation constructed tables with the follow variance matrix

$$\log_2 \mathbf{M}' = \begin{pmatrix}
28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
6.6 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
7.6 & 6.9 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
8.5 & 7.8 & 7.1 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
9.5 & 8.8 & 8.0 & 7.2 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
10.4 & 9.7 & 8.9 & 8.1 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
11.4 & 10.7 & 9.9 & 9.1 & 8.2 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
12.4 & 11.6 & 10.8 & 10.0 & 9.2 & 8.3 & 7.3 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
13.3 & 12.6 & 11.8 & 11.0 & 10.1 & 9.2 & 8.3 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
14.3 & 13.6 & 12.8 & 12.0 & 11.1 & 10.2 & 9.3 & 8.3 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
15.3 & 14.5 & 13.8 & 12.9 & 12.1 & 11.2 & 10.3 & 9.3 & 8.4 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
16.3 & 15.5 & 14.7 & 13.9 & 13.1 & 12.2 & 11.3 & 10.3 & 9.4 & 8.4 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
17.3 & 16.5 & 15.7 & 14.9 & 14.0 & 13.2 & 12.2 & 11.3 & 10.3 & 9.4 & 8.4 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
18.3 & 17.5 & 16.7 & 15.9 & 15.0 & 14.1 & 13.2 & 12.3 & 11.3 & 10.4 & 9.4 & 8.4 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
19.3 & 18.5 & 17.7 & 16.9 & 16.0 & 15.1 & 14.2 & 13.3 & 12.3 & 11.4 & 10.4 & 9.4 & 8.4 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
20.3 & 19.5 & 18.7 & 17.9 & 17.0 & 16.1 & 15.2 & 14.3 & 13.3 & 12.4 & 11.4 & 10.4 & 9.4 & 8.4 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
21.3 & 20.5 & 19.7 & 18.9 & 18.0 & 17.1 & 16.2 & 15.3 & 14.3 & 13.4 & 12.4 & 11.4 & 10.4 & 9.4 & 8.4 & 7.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
22.2 & 21.5 & 20.7 & 19.8 & 19.0 & 18.1 & 17.2 & 16.3 & 15.3 & 14.3 & 13.4 & 12.4 & 11.4 & 10.4 & 9.4 & 8.4 & 7.4 & 28.4 & 28.4 & 28.4 \\
23.2 & 22.4 & 21.7 & 20.8 & 20.0 & 19.1 & 18.2 & 17.3 & 16.3 & 15.3 & 14.4 & 13.4 & 12.4 & 11.4 & 10.4 & 9.4 & 8.4 & 7.4 & 28.4 & 28.4 \\
24.2 & 23.4 & 22.6 & 21.8 & 21.0 & 20.1 & 19.2 & 18.2 & 17.3 & 16.3 & 15.4 & 14.4 & 13.4 & 12.4 & 11.4 & 10.4 & 9.4 & 8.4 & 7.4 & 28.4
\end{pmatrix}$$

To highlight the limitations of Algorithm 4 we consider the parameters $q = 65521, n = 30, b = 3, \sigma = 1.0, p = 2^8$ and $o = 2^{29}$. Algorithm 4 predicts

$$\log_2 \mathbf{M}' = \begin{pmatrix}
28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
11.8 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
12.9 & 12.5 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
13.8 & 13.4 & 12.8 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
14.6 & 14.2 & 13.6 & 12.9 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
15.5 & 15.1 & 14.5 & 13.8 & 13.0 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
16.3 & 15.9 & 15.3 & 14.6 & 13.9 & 13.1 & 28.4 & 28.4 & 28.4 & 28.4 \\
17.2 & 16.8 & 16.2 & 15.5 & 14.8 & 13.9 & 13.1 & 28.4 & 28.4 & 28.4 \\
18.1 & 17.7 & 17.1 & 16.4 & 15.6 & 14.8 & 14.0 & 13.1 & 28.4 & 28.4 \\
19.0 & 18.6 & 18.0 & 17.3 & 16.5 & 15.7 & 14.9 & 14.0 & 13.2 & 28.4
\end{pmatrix}$$

but implementation produced tables with variance matrix

$$\log_2 \mathbf{M} = \begin{pmatrix}
28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
11.7 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
12.6 & 12.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
13.4 & 13.2 & 12.8 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
14.1 & 13.9 & 13.5 & 13.0 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
14.8 & 14.6 & 14.2 & 13.7 & 13.1 & 28.4 & 28.4 & 28.4 & 28.4 & 28.4 \\
15.4 & 15.3 & 15.0 & 14.5 & 13.9 & 13.2 & 28.4 & 28.4 & 28.4 & 28.4 \\
16.1 & 16.0 & 15.7 & 15.3 & 14.7 & 14.0 & 13.2 & 28.4 & 28.4 & 28.4 \\
16.9 & 16.7 & 16.4 & 16.0 & 15.5 & 14.9 & 14.1 & 13.3 & 28.4 & 28.4 \\
17.6 & 17.4 & 17.2 & 16.8 & 16.3 & 15.7 & 15.0 & 14.2 & 13.3 & 28.4
\end{pmatrix}$$

which means we overestimated the variance of child components in our tables T.

The main reason for this effect is that our approximation of the shrinking of variances when taking the minimum is based on the uniform distribution. However, the distributions actually governing the entries of our tables T^ℓ are not uniform, as discussed in 3. Figure 5.2 gives histograms for these distributions.

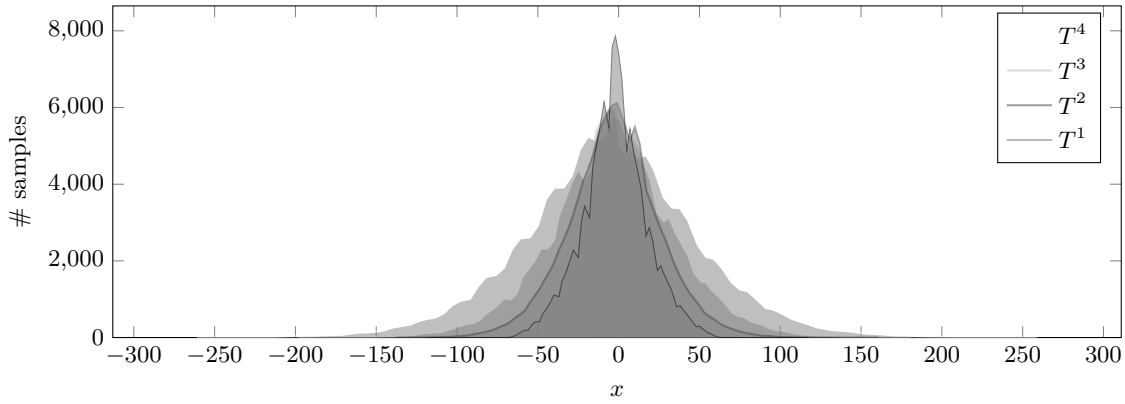

Fig. 5. Histogram of distribution of 0th component in T^ℓ for $1 \leq \ell \leq 4$ with parameters $q = 32003$, $b = 2$, $p = 2^9$, $n = 10$, and $o = 2^{20}$.

Overall, for the instances considered our estimates are pessimistic, which means we expect our algorithm to perform better in practice than predicted. A more refined model for its behaviour is hence a good topic for future work.

6 Parameters

To understand the behaviour of our more careful modulus switching technique for concrete parameters, we compare it with one-shot modulus switching. Specifically, we consider the "plain" BKW algorithm [7] as analysed in [3]. Furthmore, to make this work somewhat self-contained we also compare with the BKZ (2.0) algorithm when applied to SIS instances derived from LWE samples and with a simple meet-in-the-middle (MITM) approach or generalised birthday attack.

Instances. We choose $n \in [128, 256, 512, 1024, 2048]$ and – using [4] – pick $q \approx n^2$ and $\sigma = \frac{q}{\sqrt{2\pi n \log_2^2 n}}$ as in Regev's original encryption scheme [22]. We then consider both variants of what is known as binary-LWE in the literature: we first assume $\mathbf{s} \leftarrow_\$ \mathcal{U}(\mathbb{Z}_2^n)$ as in [8] and then $\mathbf{s} \leftarrow_\$ \mathcal{U}(\{-1, 0, 1\}^n)$ as in [12]. However, we assume an unbounded number of samples being available to the attacker to establish the performance of the algorithms discussed here under optimal conditions.

BKW. For complexity estimates of the plain BKW algorithm we rely on [3]. There the BKW algorithm takes a parameter t which controls the addition depth $a := t \log_2 n$. Here we first pick $t = 2(\log_2 q - \log_2 \sigma)/\log_2 n$ which ensures that the standard deviation of the noise after a levels of additions grows only as large as the modulus. We then slowly increase t in steps of 0.1 until the performance of the algorithm is not estimated to improve any further because too many samples are needed to perform the distinguishing step. Following [3], we translate operations in $\mathbb{Z}_q$ into "bit operations" by multiplying by $\log_2 q$.

BKZ. To estimate the cost of the BKZ (2.0) algorithm we follow [19,17]. In [19], 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 $\mathbb{Z}_q^n \times \mathbb{Z}_q$. Given a matrix of samples $\mathbf{A}$, one way of constructing such a distinguisher is to find a short vector $\mathbf{u}$ such that $\mathbf{uA} = \mathbf{0} \bmod q$. 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 $\mathbf{c} = \mathbf{A}\mathbf{s} + \mathbf{e}$, then $\langle \mathbf{u}, \mathbf{c} \rangle = \langle \mathbf{u}, \mathbf{A}\mathbf{s} + \mathbf{e} \rangle = \langle \mathbf{u}, \mathbf{e} \rangle$, 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 [19], the authors of [17] 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, σ and a target distinguishing advantage ϵ 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 $\delta_0 = 2^{\log_2^2(\beta)/(4n \log_2 q)}$.

Given δ_0 we then approximate the running time of BKZ 2.0 in seconds using two different strategies. Both strategies treat δ_0 as the dominant influence in determining the running time. The first strategy denoted "BKZ" follows [17] and defines $\log_2 T_{sec} = 1.8/\log_2 \delta_0 - 110$. The second strategy denoted "BKZ2" follows [3] who interpolated data points from [18] as $\log_2 T_{sec} = 0.009/\log_2^2 \delta_0 - 27$.

We translate the running time in seconds figure into bit operations by assuming $2.3 \cdot 10^9$ bit operations per second on a 2.3 GHz CPU, which is pessimistic. Furthermore, for BKZ choosing advantage $\epsilon \ll 1$ and running the algorithms about $1/\epsilon$ times is usually more efficient than choosing $\epsilon \approx 1$ directly, i.e. we generate a new lattice of optimal sub-dimension each time using fresh LWE samples.

MITM. One can also solve small secret LWE with a meet-in-the-middle attack that requires $\approx \mathfrak{c}^{n/2}$ time and space where $\mathfrak{c}$ is the cardinality of the set from which each component of the secret is sampled (so $\mathfrak{c} = 2$ or $\mathfrak{c} = 3$ for binary-LWE depending on the definition used): compute and store a sorted list of all $\mathbf{A}\mathbf{s}'$ where $\mathbf{s}' = (\mathbf{s}_{(0)}, \ldots, \mathbf{s}_{(n/2)-1}, 0, 0, \ldots, 0)$ for all possible $\mathfrak{c}^{n/2}$ choices for $\mathbf{s}'$. Then compute $\mathbf{c} - \mathbf{A}\mathbf{s}''$ where we have $\mathbf{s}'' = (0, 0, \ldots, 0, \mathbf{s}_{(n/2)}, \ldots, \mathbf{s}_{n-1})$ and check for a vector that is close to this value in the list.

Results for $\mathbf{s} \leftarrow_\$ \mathcal{U}(\mathbb{Z}_2^n)$. In Table 2 we give the number of bit operations ("$\log \mathbb{Z}_2$"), calls to the LWE oracle ("$\log L_{\mathbf{s},\chi}$") and memory requirement ("log mem") for BKW without any modulus reduction to establish the baseline. All costs are given for the high advantage case, i.e. if $\epsilon \ll 1$ we multiply the cost by $1/\epsilon$. Table 3 gives the running times after modulus reduction with

	MITM		BKZ [17]			BKZ 2.0 [18]			BKW [3]			
n	$\log \mathbb{Z}_2$	log mem	$\log \epsilon$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	$\log \epsilon$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	t	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	log mem
128	67.8	64	-18	26.5	65.4	-14	22.5	65.7	3.18	83.9	97.6	90.0
256	132.0	128	-29	38.5	179.5	-35	44.5	178.5	3.13	167.2	182.1	174.2
512	260.2	256	-48	58.5	390.9	-94	104.5	522.8	3.00	344.7	361.0	352.8
1024	516.3	512	-82	93.5	785.0	-265	276.5	1606.2	2.99	688.0	705.5	697.0
2048	1028.5	1024	-141	153.6	1523.6	-773	785.4	5100.0	3.00	1369.8	1388.7	1379.9

Table 2. Cost for solving Decision-LWE with advantage ≈ 1 for BKW, BKZ and MITM where q, σ are chosen as in [22] and $\mathbf{s} \leftarrow_\$ \mathcal{U}(\mathbb{Z}_2^n)$. We run BKZ $1/\epsilon$ times, "$\log \mathbb{Z}_2$" gives the number of "bit operations", "$\log L_{\mathbf{s},\chi}$" the number of LWE oracle calls and "log mem" the memory requirement of $\mathbb{Z}_q$ elements. All logarithms are base 2.

$p = q\sqrt{n/12}\sigma_s/\sigma$. In particular, Table 3 lists the expected running time (number of oracle calls and where applicable memory requirements) of running BKW and BKZ after applying modulus reduction.

Finally, Table 4 gives the expected costs for solving these LWE instances using the techniques described in this work. We list two variants: one with and one without "unnatural selection" . This is because these techniques rely on more assumptions than the rest of this work which means we have greater confidence in the predictions avoiding such assumptions. Yet, as discussed in Section 5, these assumptions seem to hold reasonably well in practice.

	BKZ [17]			BKZ 2.0 [18]			BKW [3]			
n	$\log \epsilon$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	$\log \epsilon$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	t	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	log mem
128	-20	28.3	68.5	-16	24.3	68.6	2.84	76.1	89.6	81.2
256	-31	40.3	172.5	-36	45.3	169.5	2.74	149.6	164.0	156.7
512	-49	59.3	360.2	-89	99.2	457.8	2.76	289.8	305.6	297.9
1024	-80	91.3	701.6	-232	243.2	1312.8	2.78	563.1	580.2	572.2
2048	-133	145.3	1327.6	-636	648.1	3929.5	2.71	1135.3	1153.6	1145.3

Table 3. Cost for solving Decision-LWE with advantage ≈ 1 for BKW and BKZ variants where q, σ are chosen as in [22] and $\mathbf{s} \leftarrow_\$ \mathcal{U}(\mathbb{Z}_2^n)$ after one-shot modulus reduction with $p = q\sqrt{n/12}\sigma_s/\sigma$.

	this work (w/o unnatural selection)						this work					
n	t	$\log p$	$\log m^*$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	log mem	t	$\log p$	$\log m^*$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	log mem
128	2.98	10	0	64.7	78.2	70.8	2.98	6	60	60.0	74.2	46.3
256	2.83	11	0	127.8	142.7	134.9	2.83	5	117	117.0	132.5	67.1
512	2.70	11	0	235.1	251.2	243.1	2.70	8	225	225.0	241.8	180.0
1024	2.59	12	0	477.4	494.8	486.5	2.59	10	467	467.0	485.0	407.5
2048	2.50	12	0	897.8	916.4	907.9	2.50	10	834	834.0	853.2	758.9

Table 4. Cost for solving Decision-LWE with advantage ≈ 1 with the algorithms discussed in this work when $\mathbf{s} \leftarrow_\$ \mathcal{U}(\mathbb{Z}_2^n)$.

Results for $\mathbf{s} \leftarrow_\$ \mathcal{U}(\{-1, 0, 1\}^n)$. Tables 5, 6 and 7 correspond to Tables 2, 3 and 4 and adopt the same notation.

	MITM		BKZ [17]			BKZ 2.0 [18]			BKW [3]			
n	$\log \mathbb{Z}_2$	log mem	$\log \epsilon$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	$\log \epsilon$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	t	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	log mem
128	105.2	101.4	-18	26.5	65.4	-14	22.5	65.7	3.18	83.9	97.6	90.0
256	206.9	202.9	-29	38.5	179.5	-35	44.5	178.5	3.13	167.2	182.1	174.2
512	409.9	405.8	-48	58.5	390.9	-94	104.5	522.8	3.00	344.7	361.0	352.8
1024	815.8	811.5	-82	93.5	785.0	-265	276.5	1606.2	2.99	688.0	705.5	697.0
2048	1627.5	1623.0	-141	153.6	1523.6	-773	785.4	5100.0	3.00	1369.8	1388.7	1379.9

Table 5. Cost for solving Decision-LWE with advantage ≈ 1 for BKW, BKZ and MITM where q and σ are chosen as in [22] and $\mathbf{s} \leftarrow_\$ \mathcal{U}(\{-1, 0, 1\}^n)$.

Discussion. The results in this section (summarised in Figure 6) indicate that the variants of the BKW algorithms discussed in this work compare favourably for the paramters considered. In particular, in the case $\mathbf{s} \leftarrow_\$ \mathcal{U}(\{0, 1\}^n)$ these BKW variants are the only algorithms which beat the simple MITM approach. For $\mathbf{s} \leftarrow_\$ \mathcal{U}(\{-1, 0, 1\}^n)$ that advantage naturally increases. The results in this table also indicate that the unnatural selection strategy has little impact on the overall time complexity. However, it allows to reduce the data complexity, in some cases, considerably. In particular, e.g. considering line 1 of Table 7, we note that applying this technique can make the difference between a feasible ($\approx 80 \cdot 1024^4$ bytes) and infeasible ($\approx 1260 \cdot 1024^6$ bytes) attack for a well-equipped attacker [14]. Finally, we note that our results indicate that lattice reduction benefits from modulus reduction. However, this seems somewhat implausible judging from the used algorithms. This might indicate that the lattice reduction estimates from the literature above might need to be revised.

Reproducibility. Since the results of this section rely on estimates which involve numerical approximations we make the source code (written for Sage [23]) we used to compute these figures available as an attachment to this document.

	BKZ [17]			BKZ 2.0 [18]			BKW [3]			
n	$\log \epsilon$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	$\log \epsilon$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	t	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	$\log$ mem
128	-21	29.3	70.2	-16	24.4	69.8	2.85	76.8	90.2	82.4
256	-31	40.3	175.3	-37	46.3	172.8	2.85	150.4	165.6	153.7
512	-50	60.3	365.0	-90	100.2	467.0	2.76	293.8	309.6	301.9
1024	-81	92.3	710.1	-236	247.2	1339.1	2.78	570.3	587.4	579.4
2048	-134	146.3	1342.3	-647	659.2	4006.5	2.71	1149.0	1167.3	1159.1

Table 6. Cost for solving Decision-LWE with advantage ≈ 1 for BKW and BKZ variants where q, σ are chosen as in [22] and $\mathbf{s} \leftarrow_\$ \mathcal{U}(\{-1, 0, 1\}^n)$ after one-shot modulus reduction with $p = q\sqrt{n/12}\sigma_s/\sigma$.

	this work (w/o unnatural selection)						this work					
n	t	$\log p$	$\log m^*$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	$\log$ mem	t	$\log p$	$\log m^*$	$\log L_{\mathbf{s},\chi}$	$\log \mathbb{Z}_2$	$\log$ mem
128	2.98	10	0	64.7	78.2	70.8	2.98	6	61	61.0	75.2	46.3
256	2.83	11	0	127.8	142.7	134.9	2.83	5	118	118.0	133.5	67.1
512	2.70	11	0	235.1	251.2	243.1	2.70	8	225	225.0	241.8	180.0
1024	2.59	12	0	477.4	494.8	486.5	2.59	10	467	467.0	485.0	407.5
2048	2.50	12	0	971.4	990.7	907.9	2.50	12	961	961.0	980.2	907.9

Table 7. Cost for solving Decision-LWE with advantage ≈ 1 with the algorithms discussed in this work when $\mathbf{s} \leftarrow_\$ \mathcal{U}(\{-1, 0, 1\}^n)$.

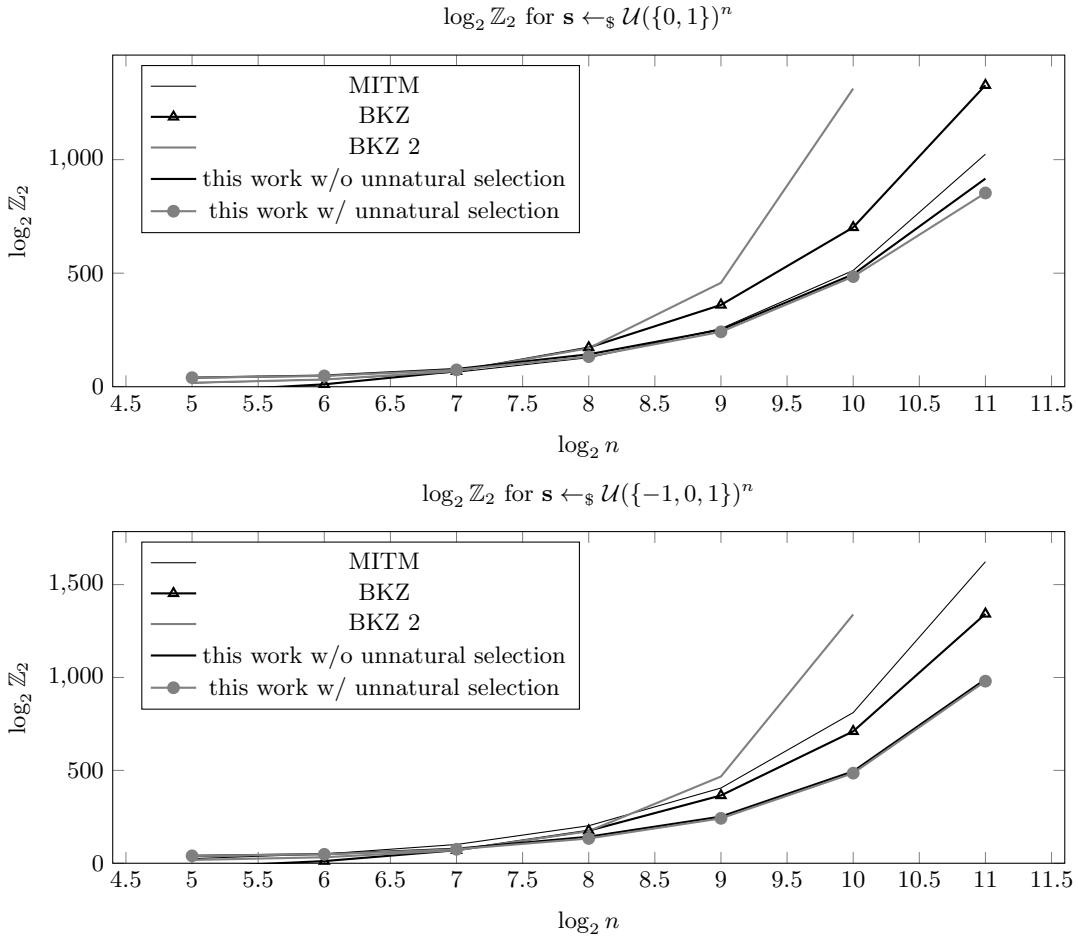

Fig. 6. Cost of solving Decision-LWE using various algorithms discussed in this work.

7 Conclusion & Future Work

We investigated applying modulus switching to exploit the presence of a small secret in LWE instances and demonstrated that it can make a significant impact on the complexity of solving such instances. We also adapted the BKW algorithm to perform modulus-switching 'on-the-fly', showing that this approach is superior to performing 'one-shot' modulus reduction on LWE samples prior to solving. Our first variant improves the target modulus by a factor of $\sqrt{\log_2 n}$ in typical scenarios; our second variant mainly improves the memory requirements of the algorithm, one of the key limiting aspects of the BKW algorithm. Our algorithms, however, rely on various assumptions which, though appearing sound, are unproven. Our estimates should thus be considered heuristic, as are performance estimates for all currently-known algorithms for solving LWE. Verifying these assumptions is hence a promising direction for future research. Furthermore, one of the main remaining obstacles for applying the BKW algorithm to cryptographic constructions based on LWE is that it requires an unbounded number of samples to proceed. Lifting this requirement, if only heuristically, is hence a pressing research question.

Acknowledgement

We thank Steven Galbraith for helpful comments on an earlier draft of this work. We also thank anonymous referees for detailed comments which greatly improved this work. Jean-Charles Faugère, and Ludovic Perret have been partially supported supported by the Computer Algebra and Cryptography (CAC) project (ANR-09-JCJCJ-0064-01) and the HPAC grant (ANR ANR-11-BS02-013) of the French National Research Agency.

References

1. Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 474–495. Springer Verlag, 2009.
2. 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.
3. Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Designs, Codes and Cryptography*, pages 1–30, 2013.
4. 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`.
5. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology – CRYPTO 2009*, Lecture Notes in Computer Science, pages 595–618, Berlin, Heidelberg, New York, 2009. Springer Verlag.
6. 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. Springer Verlag, 2011.
7. 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.
8. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of Learning with Errors. In *STOC '13*, pages 575–584, New York, 2013. ACM.
9. 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.

10. 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.
11. 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.
12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867, Berlin, Heidelberg, New York, 2012. Springer Verlag.
13. Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the Learning with Errors assumption. In Andrew Chi-Chih Yao, editor, *ICS*, pages 230–240. Tsinghua University Press, 2010.
14. Kashmir Hill. Blueprints of NSA's ridiculously expensive data center in Utah suggest it holds less info than thought. `http://www.forbes.com/sites/kashmirhill/2013/07/24/blueprints-of-nsa-data-center-in-utah-suggest-its-storage-capacity-is-less-impressive-than-thought/`, 2013.
15. Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.
16. Paul Kirchner. Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377, 2011. `http://eprint.iacr.org/`.
17. Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. *IACR Cryptology ePrint Archive*, 2010:592, 2010.
18. Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In Ed Dawson, editor, *Topics in Cryptology - CT-RSA 2013*, volume 7779 of *Lecture Notes in Computer Science*, pages 293–309. Springer Verlag, 2013.
19. 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.
20. J. J. Moré, B. S. Garbow, and K. E. Hillstrom. *User Guide for MINPACK-1*, 1980.
21. Krzysztof Pietrzak. Subspace LWE. In Ronald Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 548–563. Springer, 2012.
22. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.
23. W. A. Stein et al. *Sage Mathematics Software (Version 5.9)*. The Sage Development Team, 2013. `http://www.sagemath.org`.