Making the BKW Algorithm Practical for LWE

Alessandro Budroni[2], Qian Guo[1,2], Thomas Johansson[1], Erik Mårtensson[1] and Paul Stankovski Wagner[1]

[1] Dept. of Electrical and Information Technology, Lund University, Sweden
[2] Selmer Center, Department of Informatics, University of Bergen, Norway
`{qian.guo,thomas.johansson,erik.martensson,paul.stankovski_wagner}@eit.lth.se`
`alessandro.budroni@uib.no`

Abstract. The Learning with Errors (LWE) problem is one of the main mathematical foundations of post-quantum cryptography. One of the main groups of algorithms for solving LWE is the Blum-Kalai-Wasserman (BKW) algorithm. This paper presents new improvements for BKW-style algorithms for solving LWE instances. We target minimum concrete complexity and we introduce a new reduction step where we partially reduce the last position in an iteration and finish the reduction in the next iteration, allowing non-integer step sizes. We also introduce a new procedure in the secret recovery by mapping the problem to binary problems and applying the Fast Walsh Hadamard Transform. The complexity of the resulting algorithm compares favourably to all other previous approaches, including lattice sieving. We additionally show the steps of implementing the approach for large LWE problem instances. The core idea here is to overcome RAM limitations by using large file-based memory.

Keywords: BKW · LWE · Lattice-Based Cryptography · FWHT · Post-Quantum Cryptography

1 Introduction

Since a large-scale quantum computer easily breaks both the problem of integer factoring and the discrete logarithm problem [34], public-key cryptography needs to be based on other underlying mathematical problems. In post-quantum cryptography - the research area studying such replacements - lattice-based problems are the most promising candidates. In the NIST post-quantum standardization competition, 5 out of 7 finalists and 2 out of 8 alternates are lattice-based [1].

The Learning with Errors problem (LWE) introduced by Regev in [33], is the main problem in lattice-based cryptography. It has a theoretically very interesting average-case to worst-case reduction to standard lattice-based problems. It has many cryptographic applications, including but not limited to, design of Fully Homomorphic Encryption Schemes (FHE). An interesting special case of LWE is the Learning Parity with Noise problem (LPN), introduced in [12], which has interesting applications in light-weight cryptography.

Considerable cryptanalytic effort has been spent on algorithms for solving LWE. These can be divided into three categories: lattice-reduction, algebraic methods and combinatorial methods. The algebraic methods were introduced by Arora and Ge in [9] and further considered in [3]. For very small noise these methods perform very well, but otherwise the approach is inefficient. The methods based on lattice-reduction are currently the most efficient ones in practise. One way of comparing the different approaches is through the Darmstadt LWE Challenges [2], where the lattice-based approach called General Sieve Kernel (G6K) is the currently most successful algorithm in breaking challenges [5]. The combinatorial algorithms are all based on the Blum-Kalai-Wasserman (BKW) algorithm and algorithms in this direction will be the focus of this paper.

For surveys on the concrete and asymptotic complexity of solving LWE, see [7] and [22, 24], respectively. In essence, BKW-style algorithms have a better asymptotic performance than lattice-based approaches for parameter choices with large noise. Unlike lattice-based approaches, BKW-style algorithms pay a penalty when the number of samples is limited (like in the Darmstadt challenges).

1.1 Related Work

The BKW algorithm was originally developed as the first subexponential algorithm for solving the LPN problem [13]. In [27] the algorithm was improved, introducing new concepts like LF2 and the use of the fast Walsh-Hadamard transform (FWHT) for the distinguishing phase. A new distinguisher using subspace hypothesis testing was introduced in [19, 20].

The BKW algorithm was first applied to the LWE problem in [4]. This idea was improved in [6], where the idea of Lazy Modulus Switching (LMS) was introduced. The idea was improved in [23, 26], where [23] introduced so called coded-BKW steps. The idea of combining coded-BKW or LMS with techniques from lattice sieving [11] lead to the next improvement [21]. This combined approach was slightly improved in [22, 30]. The distinguishing part of the BKW algorithm for solving LWE was improved by using the Fast Fourier Transform (FFT) in [16]. One drawback of BKW is its high memory-usage. To remedy this, time-memory trade-offs for the BKW algorithm were recently studied in [15, 17, 18].

1.2 Contributions

In this paper we introduce a new BKW-style algorithm including the following.

- A generalized reduction step that we refer to as smooth-LMS, allowing us to use non-integer step sizes. These steps allow us to use the same time, space and sample complexity in each reduction step of the algorithm, which improves performance compared to previous work.
- A binary-oriented method for the guessing phase, transforming the LWE problem into an LPN problem. While the previous FFT method guesses a few positions of the secret vector and finds the correct one, this approach instead finds the least significant bits of a large amount of positions using the FWHT. This method allows us to correctly distinguish the secret with a larger noise level, generally leading to an improved performance compared to the FFT based method. In addition, the FWHT is much faster in implementation.

– Concrete complexity calculations for the proposed algorithm showing the lowest known complexity for some parameter choices selected as in the Darmstadt LWE Challenge instances, but with unrestricted number of samples.
– An implementation approach for the algorithm that allows larger instances to be solved. The implementation is file-based and stores huge tables on disk and not in RAM only. The file read/write is minimized by implementing the algorithm in a clever way. Simulation results on solving larger instances are presented and verifies the previous theoretical arguments.

1.3 Organization

We organize the rest of the paper as follows. We introduce some necessary background in Section 2. In Section 3 we cover previous work on applying the BKW algorithm to the LWE problem. Then in Section 4 we introduce our new Smooth-LMS reduction method. Next, in Section 5 we go over our new binary-oriented guessing procedure. Sections 6 and 7 cover the complexity analysis and implementation of our algorithm, respectively. Section 8 describes our experimental results using the implementation. Finally, the paper is concluded in Section 9.

2 Background

2.1 Notation

Throughout the paper we use the following notations.

– We write $\log(\cdot)$ for the base 2 logarithm.
– In the n-dimensional Euclidean space $\mathbb{R}^n$, by the norm of a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ we consider its L_2-norm, defined as

$$\|\mathbf{x}\| = \sqrt{x_1^2 + \cdots + x_n^2}.$$

  The Euclidean distance between vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$ is defined as $\|\mathbf{x} - \mathbf{y}\|$.
– Elements in $\mathbb{Z}_q$ are represented by the set of integers in $[-\frac{q-1}{2}, \frac{q-1}{2}]$.
– For an $[N, k]$ linear code, N denotes the code length and k denotes the dimension.

2.2 The LWE and LPN Problems

The LWE problem [33] is defined as follows.

Definition 1 *Let n be a positive integer, q a prime, and let $\mathcal{X}$ be an error distribution selected as the discrete Gaussian distribution on $\mathbb{Z}_q$ with variance σ^2. Fix $\mathbf{s}$ to be a secret vector in $\mathbb{Z}_q^n$, chosen from some distribution (usually the uniform distribution). Denote by $L_{\mathbf{s},\mathcal{X}}$ 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 an error $e \in \mathbb{Z}_q$ from $\mathcal{X}$ and returning*

$$(\mathbf{a}, z) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$$

in $\mathbb{Z}_q^n \times \mathbb{Z}_q$. The (search) LWE *problem is to find the secret vector* $\mathbf{s}$ *given a fixed number of samples from $L_{\mathbf{s},\mathcal{X}}$.*

The definition above gives the *search* LWE problem, as the problem description asks for the recovery of the secret vector $\mathbf{s}$. Another version is the *decision* LWE problem, in which case the problem is to distinguish between samples drawn from $L_{\mathbf{s},\mathcal{X}}$ and a uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Let us also define the LPN problem, which is a binary special case of LWE.

Definition 2 *Let k be a positive integer, let $\mathbf{x}$ be a secret binary vector of length k and let $X \sim \mathsf{Ber}_\eta$ be a Bernoulli distributed error with parameter $\eta > 0$. Let $L_{\mathbf{x},X}$ denote the probability distribution on $\mathbb{F}_2^k \times \mathbb{F}_2$ obtained by choosing* $\mathbf{g}$ *uniformly at random, choosing $e \in \mathbf{F}_2$ from X and returning*

$$(\mathbf{g}, z) = (\mathbf{g}, \langle \mathbf{g}, \mathbf{x} \rangle + e)$$

The (search) LPN *problem is to find the secret vector* $\mathbf{s}$ *given a fixed number of samples from $L_{\mathbf{x},X}$.*

Just like for LWE, we can also, analogously, define *decision* LPN.

Previously, analysis of algorithms solving the LWE problem have used two different approaches. One being calculating the number of operations needed to solve a certain instance for a particular algorithm, and then comparing the different complexity results. The other being asymptotic analysis. Solvers for the LWE problem with suitable parameters are expected to have fully exponential complexity, bounded by 2^{cn} as n tends to infinity, where the value of c depends on the algorithms and the parameters of the involved distributions. In this paper, we focus on the complexity computed as the number of arithmetic operations in $\mathbb{Z}_q$, for solving particular LWE instances (and we do not consider the asymptotics).

2.3 Discrete Gaussian Distributions

We define the discrete Gaussian distribution over $\mathbb{Z}$ with mean 0 and variance σ^2, denoted $D_{\mathbb{Z},\sigma}$ as the probability distribution obtained by assigning a probability proportional to $\exp(-x^2/(2\sigma^2))$ to each $x \in \mathbb{Z}$. Then, the discrete Gaussian distribution $\mathcal{X}$ over $\mathbb{Z}_q$ with variance σ^2 (also denoted $\mathcal{X}_\sigma$) can be defined by folding $D_{\mathbb{Z},\sigma}$ and accumulating the value of the probability mass function over all integers in each residue class modulo q. It makes sense to consider the noise level as α, where $\sigma = \alpha q$. We also define the rounded Gaussian distribution on Z_q. This distribution samples values by sampling values from the continuous Gaussian distribution with mean 0 and variance σ^2, rounding to the closest integer and then folding the result to the corresponding value in $\mathbb{Z}_q$. We denote it by $\bar{\Psi}_{\sigma,q}$.

If two independent X_1 and X_2 are drawn from $\mathcal{X}_{\sigma_1}$ and $\mathcal{X}_{\sigma_2}$ respectively, we make the heuristic assumption that their sum is drawn from $\mathcal{X}_{\sqrt{\sigma_1^2+\sigma_2^2}}$. We make the corresponding assumption for the rounded Gaussian distribution.

3 A Review of BKW-style Algorithms

3.1 The LWE Problem Reformulated

Assume that m samples

$$(\mathbf{a}_1, z_1), (\mathbf{a}_2, z_2), \ldots, (\mathbf{a}_m, z_m),$$

are collected from the LWE distribution $L_{\mathbf{s},\mathcal{X}}$, where $\mathbf{a}_i \in \mathbb{Z}_q^n, z_i \in \mathbb{Z}_q$. Let $\mathbf{z} = (z_1, z_2, \ldots, z_m)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_m) = \mathbf{sA}$. We have

$$\mathbf{z} = \mathbf{sA} + \mathbf{e},$$

where $\mathbf{A} = \left[\mathbf{a}_1^{\mathrm{T}}\, \mathbf{a}_2^{\mathrm{T}} \cdots \mathbf{a}_m^{\mathrm{T}}\right]$, $z_i = y_i + e_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i$ and $e_i \overset{\$}{\leftarrow} \mathcal{X}$. The search LWE problem is a decoding problem, where $\mathbf{A}$ serves as the generator matrix for a linear code over $\mathbb{Z}_q$ and $\mathbf{z}$ is a received word. Finding the secret vector $\mathbf{s}$ is equivalent to finding the codeword $\mathbf{y} = \mathbf{sA}$ for which the Euclidean distance $||\mathbf{y} - \mathbf{z}||$ is minimal. In the sequel, we adopt the notation $\mathbf{a}_i = (a_{i1}, a_{i2}, \ldots, a_{in})$.

3.2 Transforming the Secret Distribution

A transformation [8,25] can be applied to ensure that the secret vector follows the same distribution $\mathcal{X}$ as the noise. It is done as follows. We write $\mathbf{A}$ in systematic form via Gaussian elimination. Assume that the first n columns are linearly independent and form the matrix $\mathbf{A_0}$. Define $\mathbf{D} = \mathbf{A_0}^{-1}$ and write $\hat{\mathbf{s}} = \mathbf{s}\mathbf{D}^{-1} - (z_1, z_2, \ldots, z_n)$. Hence, we can derive an equivalent problem described by $\hat{\mathbf{A}} = (\mathbf{I}, \hat{\mathbf{a}}_{n+1}^{\mathrm{T}}, \hat{\mathbf{a}}_{n+2}^{\mathrm{T}}, \cdots, \hat{\mathbf{a}}_m^{\mathrm{T}})$, where $\hat{\mathbf{A}} = \mathbf{DA}$. We compute

$$\hat{\mathbf{z}} = \mathbf{z} - (z_1, z_2, \ldots, z_n)\hat{\mathbf{A}} = (\mathbf{0}, \hat{z}_{n+1}, \hat{z}_{n+2}, \ldots, \hat{z}_m).$$

Using this transformation, each entry in the secret vector $\mathbf{s}$ is now distributed according to $\mathcal{X}$. The fact that entries in $\mathbf{s}$ are small is a very useful property in several of the known reduction algorithms for solving LWE.

The noise distribution $\mathcal{X}$ is usually chosen as the discrete Gaussian distribution or the rounded Gaussian Distribution from Section 2.3.

3.3 Sample Amplification

In some versions of the LWE problem, such as the Darmstadt Challenges [2], the number of available samples is limited. To get more samples, sample amplification can be used. For example, assume that we have M samples $(\mathbf{a}_1, b_1)$, $(\mathbf{a}_2, b_2)$, ..., $(\mathbf{a}_M, b_M)$. Then we can form new samples, using an index set I of size k, as

$$\left(\sum_{j \in I} \pm \mathbf{a}_j, \sum_{j \in I} \pm b_j \right). \tag{1}$$

Given an initial number of samples M we can produce up to $2^{k-1}\binom{M}{k}$ samples. This comes at a cost of increasing the noise level (standard deviation) to $\sqrt{k} \cdot \sigma$. This also increases the sample dependency.

3.4 Iterating and Guessing

BKW-style algorithms work by combining samples in many steps in such a way that we reach a system of equations over $\mathbb{Z}_q$ of the form $\mathbf{z} = \mathbf{sA} + \mathbf{E}$, where $\mathbf{E} = (E_1, E_2, \ldots, E_m)$ and the entries $E_i, i = 1, 2, \ldots, m$ are sums of not too many original noise vectors, say $E_i = \sum_{j=1}^{2^t} e_{i_j}$, and where t is the number of iterations. The process also reduces the norm of column vectors in $\mathbf{A}$ to be small. Let $n_i, i = 1, 2, \ldots, t$ denote the number of reduced positions in step i and let $N_i = \sum_{j=1}^{i} n_j$. If $n = N_t$, then every reduced equation is of form

$$z_i = \langle \mathbf{a_i}, \mathbf{s} \rangle + E_i, \tag{2}$$

for $i = 1, 2, \ldots, m$. The right hand side can be approximated as a sample drawn from a discrete Gaussian and if the standard deviation is not too large, then the sequence of samples $z_1, z_2, \ldots$ can be distinguished from a uniform distribution. We will then need to determine the number of required samples to distinguish between the uniform distribution on $\mathbb{Z}_q$ and $\mathcal{X}_\sigma$. Relying on standard theory from statistics, using either previous work [28] or Bleichenbacher's definition of bias [32], we can find that the required number of samples is roughly

$$C \cdot e^{2\pi \left(\frac{\sigma \sqrt{2\pi}}{q} \right)^2}, \tag{3}$$

where C is a small positive constant. Initially, an optimal but exhaustive distinguisher was used [10]. While minimizing the sample complexity, it was slow and limited the number of positions that could be guessed. This basic approach was improved in [16], using the FFT. This was in turn a generalization of the corresponding distinguisher for LPN, which used the FWHT [27].

3.5 Plain BKW

The basic BKW algorithm was originally developed for solving LPN in [13]. It was first applied to LWE in [4]. The reduction part of this approach means that we reduce a fixed number b of positions in the column vectors of $\mathbf{A}$ to zero in each step. In each iteration, the dimension of $\mathbf{A}$ is decreased by b and after t iterations the dimension has decreased by bt.

3.6 Coded-BKW and LMS

LMS was introduced in [6] and improved in [26]. Coded-BKW was introduced in [23]. Both methods reduce positions in the columns of $\mathbf{A}$ to a small magnitude, but not to zero, allowing reduction of more positions per step. In LMS this is achieved by mapping samples to the same category if the n_i considered positions give the same result when integer divided by a suitable parameter p. In coded-BKW this is instead achieved by mapping samples to the same category if they are close to the same codeword in an $[n_i, k_i]$ linear code, for a suitable value k_i. Samples mapped to the same category give rise to new samples by subtracting them. The main idea [23, 26] is that positions in later iterations do not need to be reduced as much as the first ones, giving different n_i values in different steps.

3.7 LF1, LF2, Unnatural Selection

Each step of the reduction part of the BKW algorithm consists of two parts. First samples are mapped to categories depending on their position values on the currently relevant n_i positions. Next, pairs of samples within the categories are added/subtracted to reduce the current n_i positions to form a new generation of samples. This can be done in a couple of different ways.

Originally this was done using what is called LF1. Here we pick a representative from each category and form new samples by adding/subtracting samples to/from this sample. This approach makes the final samples independent, but also gradually decreases the sample size. In [27] the approach called LF2 was introduced. Here we add/subtract every possible pair within each category to form new samples. This approach requires only 3 samples within each category to form a new generation of the same size. The final samples are no longer independent, but experiments have shown that this effect is negligible.

In [6] unnatural selection was introduced.The idea is to produce more samples than needed from each category, but only keep the best samples, typically the ones with minimum norm on the current N_i positions in the columns of $\mathbf{A}$.

3.8 Coded-BKW with Sieving

When using coded-BKW or LMS, the previously reduced N_{i-1} positions of the columns of $\mathbf{A}$ increase in magnitude with an average factor $\sqrt{2}$ in each reduction step. This problem was addressed in [21] by using unnatural selection to only produce samples that kept the magnitude of the previous N_{i-1} positions small. Instead of testing all possible pairs of samples within the categories, this procedure was sped-up using lattice sieving techniques of [11]. This approach was slightly improved in [22, 30].

4 BKW-style Reduction Using Smooth-LMS

In this section we introduce a new reduction algorithm solving the problem of having the same complexity and memory usage in each iteration of a BKW-style reduction. The novel idea is to use simple LMS to reduce a certain number of positions and then partially reduce one extra position. This allows for balancing the complexity among the steps and hence to reduce more positions in total.

4.1 A New BKW-style Step

Assume having a large set of samples written as before in the form $\mathbf{z} = \mathbf{sA} + \mathbf{e} \bmod q$. Assume also that the entries of the secret vector $\mathbf{s}$ are drawn from some restricted distribution with small standard deviation (compared to the alphabet size q). If this is not the case, the transformation from Section 3.2 should be applied. Moreover, in case the later distinguishing process involves some positions to be guessed or transformed, we assume that this has been already considered and all positions in our coming description should be reduced.

The goal of this BKW-type procedure is to make the norms of the column vectors of $\mathbf{A}$ small by adding and subtracting equations together in a number of steps. Having expressions of the form $z_i = \mathbf{s}\mathbf{a}_i + E_i \bmod q$, if we can reach a case where $||\mathbf{a}_i||$ is not too large, then $\mathbf{s}\mathbf{a}_i + E_i$ can be considered as a random variable drawn from a discrete Gaussian distribution $\mathcal{X}_\sigma$. Furthermore, $\mathcal{X}_\sigma \bmod q$ can be distinguished from a uniform distribution over $\mathbb{Z}_q$ if σ is not too large.

Now let us describe the new reduction procedure. Fix the number of reduction steps to be t. We will also fix a maximum list size to be 2^v, meaning that $\mathbf{A}$ can have at most 2^v columns. In each iteration i, we are going to reduce some positions to be upper limited in magnitude by C_i, for $i = 1, ..., t$. Namely, these positions that are fully treated in iteration i will only have values in the set $\{-C_i + 1, \ldots, 0, 1, \ldots, C_i - 1\}$ of size $2C_i - 1$. We do this by dividing up the q possible values into intervals of length C_i. We also adopt the notation $\beta_i = q/C_i$, which describes the number of intervals we divide up the positions into. We assume that $\beta_i > 2$.

First step. In the first iteration, assume that we have stored $\mathbf{A}$. We first compute the required compression starting in iteration 1 by computing C_1 (we will explain how later). We then evaluate how many positions n_1 that can be fully reduced by computing $n_1 = \lfloor v/\log \beta_1 \rfloor$. The position $n_1 + 1$ can be partially reduced to be in an interval of size C_1' fulfilling $\beta_1' \cdot \beta_1^{n_1} \cdot 3/2 \leq 2^v$, where $\beta_1' = q/C_1'$. Now we do an LMS step that "transfers between iterations" in the following way.

We run through all the columns of $\mathbf{A}$. For column i, we simply denote it as $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and we compute:

$$k_j = \begin{cases} x_j \text{ div } C_1, & x_1 \geq 0 \\ -x_j \text{ div } C_1, & x_1 < 0 \end{cases}, \quad \text{for} \quad j = 1, \ldots, n_1,$$

$$k_{n_1+1} = \begin{cases} x_{n_1+1} \text{ div } C_1', & x_1 \geq 0 \\ -x_{n_1+1} \text{ div } C_1', & x_1 < 0 \end{cases}.$$

The vector $K_i = (k_1, k_2, \ldots, k_{n_1+1})$ is now an index to a sorted list $\mathcal{L}$, storing these vectors[3]. Except for the inverting of values if $x_1 < 0$, samples should have the same index if and only if all position values are the same when integer divided by C_1 (C_1' for the last position). So we assign $\mathcal{L}(K_i) = \mathcal{L}(K_i) \cup \{i\}$. After we have inserted all columns into the list $\mathcal{L}$, we go to the combining part.

We build a new matrix $\mathbf{A}$ in the following way. Run through all indices K and if $|\mathcal{L}(K)| \geq 2$ combine every pair of vectors in $\mathcal{L}(K)$ by subtracting/adding[4] them to form a new column in the new matrix $\mathbf{A}$. Stop when the number of new columns has reached 2^v. For each column in $\mathbf{A}$ we have that:

- the absolute value of each position $j \in \{1, \ldots, n_1\}$ is $< C_1$,
- the absolute value of position $n_1 + 1$ is $< C_1'$.

[3] The point of inverting all position values if $x_1 < 0$ is to make sure that samples that get reduced when added should be given the same index. For example $(x_1, x_2, \ldots, x_{n_1+1})$ and $(-x_1, -x_2, \ldots, -x_{n_1+1})$ are mapped to the same category.

[4] Depending on what reduces the sample the most.

Next steps. We now describe all the next iterations, numbered as $l = 2, 3, \ldots, t$. Iteration l will involve positions from $N_{l-1}+1 = \sum_{i=1}^{l-1} n_i + 1$ to $N_l + 1$. The very first position has possibly already been partially reduced and its absolute value is $< C'_{l-1}$, so the interval for possible values is of size $2C'_{l-1} - 1$. Assume that the desired interval size in iteration l is C_l. In order to achieve the corresponding reduction factor β_l, we split this interval in $\beta''_l = (2C'_{l-1} - 1)/C_l$ subintervals. We then compute how many positions n_l that can be fully reduced by computing $n_l = \lfloor (v - \log \beta''_l)/\log \beta_l \rfloor$. The position $N_l + 1$ can finally be partially reduced to be in an interval of size C'_l fulfilling $\beta'_l \cdot \beta_l^{n_l - 1} \beta''_l \cdot 3/2 \leq 2^v$, where $\beta'_l = q/C'_l$.

Similar to iteration 1, we run through all the columns of $\mathbf{A}$. For each column i in the matrix $\mathbf{A}$ denoted as $\mathbf{x}$ we do the following. For each vector position in $\{N_{l-1}+1, \ldots, N_l + 1\}$, we compute (here div means integer division)

$$k_j = \begin{cases} x_{N_{l-1}+j} \text{ div } C_l, & x_{N_{l-1}+1} \geq 0 \\ -x_{N_{l-1}+j} \text{ div } C_l, & x_{N_{l-1}+1} < 0 \end{cases}, \quad \text{for} \quad j = 1, \ldots, n_l,$$

$$k_{n_l} = \begin{cases} x_{N_l+1} \text{ div } C'_l, & x_{N_{l-1}+1} \geq 0 \\ -x_{N_l+1} \text{ div } C'_l, & x_{N_{l-1}+1} < 0 \end{cases}. \tag{4}$$

The vector $K = (k_1, k_2, \ldots, k_{n_l+1})$ is again an index to a sorted list $\mathcal{L}$, keeping track of columns[5]. So again we assign $\mathcal{L}(K) = \mathcal{L}(K) \cup \{i\}$. After we have inserted all column indices into the list $\mathcal{L}$, we go to the combining part.

As in the first step, we build a new $\mathbf{A}$ as follows. Run through all indices K and if $|\mathcal{L}(K)| \geq 2$ combine every pair of vectors by adding/subtracting them to form a column in the new matrix $\mathbf{A}$. Stop when the number of new columns has reached 2^v.

For the last iteration, since N_t is the last row of $\mathbf{A}$, one applies the same step as above but without reducing the extra position. After t iterations, one gets equations on the form (2), where the $\mathbf{a}_i$ vectors in $\mathbf{A}$ have reduced norm.

4.2 Smooth-Plain BKW

The procedure described above also applies to plain BKW steps. For example, if in the first iteration one sets $C_1 = 1$ and $C'_1 > 1$, then each column vector $\mathbf{x}$ of $\mathbf{A}$ will be reduced such that $x_1 = \ldots = x_{n_1} = 0$ and $x_{n_1+1} \in \{-C'_1+1, \ldots, C'_1 - 1\}$. Thus, one can either continue with another smooth-Plain BKW step by setting also $C_2 = 1$ in the second iteration, or switch to smooth-LMS. In both cases, we have the advantage of having x_{n_1} already partially reduced. Using these smooth-Plain steps we can reduce a couple of extra positions in the plain pre-processing steps of the BKW algorithm.

[5] Also here the point of inverting all position values if $x_{N_{l-1}+1} < 0$ is to make sure that samples that get reduced when added should be given the same index. For example $(x_{N_{l-1}+1}, x_{N_{l-1}+2}, \ldots, x_{N_l+1})$ and $(-x_{N_{l-1}+1}, -x_{N_{l-1}+2}, \ldots, -x_{N_l+1})$ are mapped to the same category.

4.3 How to Choose the Interval Sizes C_i

To achieve as small norm of the vectors as possible, we would like the variance of all positions to be equally large, after completing all iterations. Assume that a position x takes values uniformly in the set $\{-(C-1)/2, \ldots, 0, 1, \ldots, (C-1)/2\}$, for $C > 0$. Then, we have that in $\mathrm{Var}[x] = (C-1)(C+1)/12$. Assuming C is somewhat large, we approximately get $\mathrm{Var}[x] = C^2/12$. When subtracting/adding two such values, the variance increases to $2\mathrm{Var}[x]$ in each iteration. Therefore, a reduced position will have an expected growth of $\sqrt{2}$. For this reason, we choose a relation for the interval sizes of the form

$$C_i = 2^{-(t-i)/2} C_t, \quad i = 1, \ldots, t-1.$$

This makes the variance of each position roughly the same, after completing all iterations. In particular, our vectors $||\mathbf{a}_i||$ in $\mathbf{A}$ are expected to have norm at most $\sqrt{n} C_t / \sqrt{12}$, and C_t is determined according to the final noise allowed in the guessing phase. Ignoring the pre-processing step with smooth-Plain BKW steps, the maximum dimension n that can be reduced is then $n = N_t = \sum_{i=1}^{t} n_i$.

Example 1. Let $q = 1601$ and $\alpha = 0.005$, so $\sigma = \alpha q \approx 8$. Let us compute how many positions that can be reduced using $2^v = 2^{28}$ list entries. The idea is that the variance of the right hand side in (2) should be minimized by making the variance of the two terms roughly equal. The error part E_i is the sum of 2^t initial errors, so its variance is $\mathrm{Var}[E_i] = 2^t \sigma^2$. In order to be able to distinguish the samples according to (3), we set $\mathrm{Var}[E_i] < q^2/2$. This will give us the number of iterations possible as $2^t \sigma^2 \approx q^2/2$ or $2^t \approx 1601^2/(2 \cdot 8^2)$ leading to $t = 14$. Now we bound the variance of the scalar product part of (2) also to be $< q^2/2$, so $n\sigma^2 C_t^2/12 \approx q^2/2$ leading to $C_t^2 \approx 12q^2/(2n\sigma^2)$ and $C_t^2 \approx 12 \cdot 1601^2/(2n \cdot 8^2)$ or $C_t \approx 80$ if $n < 38$. Then one chooses $C_{t-1} = \lfloor C_t/\sqrt{2} \rceil = 57$ and so on.

4.4 Unnatural Selection

We can improve performance by using the unnatural selection discussed in Section 3.7. Let us make some basic observations. Combining n_l positions using interval size C gives as previously described a value in the set $\{-(C-1)/2, \ldots, 0, 1, \ldots (C-1)/2\}$, and results in $\mathrm{Var}[x] = (C-1)(C+1)/12$. Combining two vectors from the same category, a position value $y = x_1 + x_2$, where x_1, x_2 are as above, results in a value in the interval $\{-(C-1), \ldots, 0, 1, \ldots (C-1)\}$ with variance $\mathrm{Var}[y] = (C-1)(C+1)/6$. Now observe that for the resulting reduced positions, smaller values are much more probable than larger ones.

5 A Binary Partial Guessing Approach

In this section we propose a new way of reducing the guessing step to a binary version. This way, we are able to efficiently use the FWHT to guess many entries in a small number of operations. In Section 6 we do the theoretical analysis and show that this indeed leads to a more efficient procedure than all previous ones.

5.1 From LWE to LPN

First, we need to introduce a slight modification to the original system of equations *before* the reduction part. Assume that we have turned the distribution of $\mathbf{s}$ to be the noise distribution, through the standard transformation described in Section 3.2. The result after this is written as before

$$\mathbf{z} = \mathbf{s}\mathbf{A} + \mathbf{e}. \tag{5}$$

Now we perform a multiplication by 2 to each equation, resulting in

$$\mathbf{z}' = \mathbf{s}\mathbf{A}' + 2\mathbf{e},$$

since when multiplied with a known value, we can compute the result modulo q.

Next, we apply the reduction steps and make the values in $\mathbf{A}'$ as small as possible by performing BKW-like steps. In our case we apply the smooth-LMS step from the previous section, but any other reduction method like coded-BKW with sieving would be possible. If $\mathbf{A}' = \begin{bmatrix}\mathbf{a}_1^{\mathrm{T}} \; \mathbf{a}_2^{\mathrm{T}} \; \cdots \; \mathbf{a}_m^{\mathrm{T}}\end{bmatrix}$ the output of this step is a matrix where the Euclidean norm of each $\mathbf{a}_i$ is small. The result is written as

$$\mathbf{z}'' = \mathbf{s}\mathbf{A}'' + 2\mathbf{E}, \tag{6}$$

where $\mathbf{E} = (E_1, E_2, \ldots, E_m)$ and $E_i = \sum_{j=1}^{2^t} e_{i_j}$ as before.

Finally, we transform the entire system to the binary case by considering

$$\mathbf{z}_0'' = \mathbf{s}_0\mathbf{A}_\mathbf{0}'' + \mathbf{e} \bmod 2, \tag{7}$$

where $\mathbf{z}_0''$ is the vector of least significant bits in $\mathbf{z}''$, $\mathbf{s}_0$ the vector of least significant bits in $\mathbf{s}$, $\mathbf{A}_\mathbf{0}'' = (\mathbf{A}'' \bmod 2)$ and $\mathbf{e}$ denotes the binary error introduced.

We can now examine the error e_j in position j of $\mathbf{e}$. In (6) we have equations of the form $z_j = \sum_i s_i a_{ij} + 2E_j$ in $\mathbb{Z}_q$, which can be written on integer form as

$$z_j = \sum_i s_i a_{ij} + 2E_j + k_j \cdot q. \tag{8}$$

Now if $|\sum_i s_i a_{ij} + 2E_j| < q/2$ then $k_j = 0$. In this case (8) can be reduced mod 2 without error and $e_j = 0$. In general, the error is computed as $e_j = k_j \bmod 2$. So one can compute a distribution for $e_j = k_j \bmod 2$ by computing $P(k_j = x)$. It is possible to compute such distribution either making a general approximation or precisely for each specific position j using the known values $\mathbf{a}_j$ and z_j. Note that the distribution of e_j depends on z_j. Also note that if $\mathbf{a}_j$ is reduced to a small norm and the number of steps t is not too large, then it is quite likely that $|\sum_i s_i a_{ij} + 2E_j| < q/2$ leading to $P(e_j = 0)$ being large.

For the binary system, we finally need to find the secret value $\mathbf{s}_0$. Either

1. there are no errors (or almost no errors), corresponding to $P(e_j = 0) \approx 1$. Then one can solve for $\mathbf{s}_0$ directly using Gaussian elimination (or possibly some information set decoding algorithm in the case of a few possible errors).
2. or the noise is larger. The binary system of equations corresponds to the situation of a fast correlation attack [31], or secret recovery in an LPN problem [13]. Thus, one may apply an FWHT to recover the binary secret values.

5.2 Guessing $\mathbf{s}_0$ Using the FWHT

The approach of using the FWHT to find the most likely $\mathbf{s}_0$ in the binary system in (7) comes directly from previous literature on Fast Correlation Attacks [14].

Let k denote an n-bit vector $(k_0, k_1, \ldots, k_{n-1})$ (also considered as an integer) and consider a sequence $X_k, k = 0, 1, \ldots, N-1$, $N = 2^n$. It can for example be a sequence of occurrence values in the time domain, e.g. $X_k =$ the number of occurrences of $X = k$. The Walsh-Hadamard Transform is defined as

$$\hat{X}_w = \sum_{k=0}^{N-1} X_k \cdot (-1)^{w \cdot k},$$

where $w \cdot k$ denotes the bitwise dot product of the binary representation of the n-bit indices w and k. There exists an efficient method (FWHT) to compute the WHT in time $O(N \log N)$. Given the matrix $\mathbf{A}''_{\mathbf{0}}$, we define $X_k = \sum_{j \in J} (-1)^{z''_j}$, where J is the set of all columns of the matrix $\mathbf{A}''_{\mathbf{0}}$ that equal k. Then, one computes $\max_w |\hat{X}_w|$, and we have that $\mathbf{s}_0$ corresponds to $\bar{w}$ such that $|\hat{X}_{\bar{w}}| = \max_w |\hat{X}_w|$. In addition, $\hat{X}_w$ is simply the (biased) sum of the noise terms.

Soft Received Information The bias of $\hat{X}_w$ actually depends on the value of z''_j. So a slightly better approach is to use "soft received information" by defining $X_k = \sum_{j \in J} (-1)^{z''_j} \cdot \epsilon_{z''_j}$, where $\epsilon_{z''_j}$ is the bias corresponding to z''_j. For each $x \in \{-(q-1)/2, ..., (q-1)/2\}$, the bias ϵ_x can be efficiently pre-computed so that its evaluation does not affect the overall complexity of the guessing procedure.

Hybrid Guessing One can use hybrid approach to balance the overall complexity among reduction and guessing phases. Indeed, it is possible to leave some rows of the matrix $\mathbf{A}$ unreduced and apply an exhaustive search over the corresponding positions in combination with the previously described guessing step.

5.3 Retrieving the Original Secret

Once $\mathbf{s}_0$ is correctly guessed, it is possible to obtain a new LWE problem instance with the secret half as big as follows. Write $\mathbf{s} = 2\mathbf{s}' + \mathbf{s}_0$. Define $\hat{\mathbf{A}} = 2\mathbf{A}$ and $\hat{\mathbf{z}} = \mathbf{z} - \mathbf{s}_0\mathbf{A}$. Then we have that

$$\hat{\mathbf{z}} = \mathbf{s}'\hat{\mathbf{A}} + \mathbf{e}. \tag{9}$$

The entries of $\mathbf{s}'$ have a bit-size half as large as the entries of $\mathbf{s}$, therefore (9) is an easier problem than (5). One can apply the procedure described above to (9) and guess the new binary secret $\mathbf{s}_1$, i.e. the least significant bits of $\mathbf{s}'$. The cost of doing this will be significantly smaller as shorter secret translates to computationally easier reduction steps. Thus, computationally speaking, the LWE problem can be considered solved once we manage to guess the least significant bits of $\mathbf{s}$. Given the list of binary vectors $\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, ...$, it is easy to retrieve the original secret $\mathbf{s}$.

Algorithm 1 BKW-FWHT with smooth reduction (main framework)

Input: Matrix $\mathbf{A}$ with n rows and m columns, received vector $\mathbf{z}$ of length m and algorithm parameters $t_1, t_2, t_3, n_{limit}, \sigma_{set}$

```
Step 0: Use Gaussian elimination to change the distribution of the secret vector;
Step 1: Use $t_1$ smooth-plain BKW steps to remove the bottom $n_{pbkw}$ entries;
Step 2: Use $t_2$ smooth-LMS steps to reduce $n_{cod1}$ more entries;
Step 3: Perform the multiplying-2 operations;
Step 4: Use $t_3$ smooth-LMS steps to reduce the remaining $n_t \leq n_{limit}$ entries;
Step 5: Transform all the samples to the binary field and recover the partial secret
    key by the FWHT. We can exhaustively guess some positions.
```

6 Analysis of the Algorithm and its Complexity

In this section, we describe in detail the newly-proposed algorithm called BKW-FWHT with smooth reduction (BKW-FWHT-SR).

6.1 The Algorithm

The main steps of the new BKW-FWHT-SR algorithm are described in Algorithm 1. We start by changing the distribution of the secret vector with the secret-noise transformation [8], if necessary.

The general framework is similar to the coded-BKW with sieving procedure proposed in [21]. In our implementation, we instantiated coded-BKW with sieving steps with smooth-LMS steps discussed before for the ease of implementation.

The different part of the new algorithm is that after certain reduction steps, we perform a multiplication by 2 to each reduced sample as described in Section 5. We then continue reducing the remain positions and perform the mod 2 operations to transform the entire system to the binary case. Now we obtain a list of LPN samples and solve the corresponding LPN instance via known techniques such as FWHT or partial guessing.

One high level description is that we aim to input an LWE instance to the LWE-to-LPN transform developed in Section 5, and solve the instance by using a solver for LPN. To optimize the performance, we first perform some reduction steps to have a new LWE instance with reduced dimension but larger noise. We then feed the obtained instance to the LWE-to-LPN transform.

6.2 The Complexity of Each Step

From now on we assume that the secret is already distributed as the noise distribution or that the secret-noise transform is performed. We use the LF2 heuristics and assume the the sample size is unchanged before and after each reduction step. We now start with smooth-plain BKW steps and let l_{red} be the number of positions already reduced.

Smooth-Plain BKW steps. Given m initial samples, we could on average have $\lfloor \frac{2m}{3} \rfloor$ categories[6] for one plain BKW step in the LF2 setting. Instead we could assume for 2^{b_0} categories, and thus the number of samples m is $1.5 \cdot 2^{b_0}$. Let C_{pBKW} be the cost of all smooth-plain BKW steps, whose initial value is set to be 0. If a step starts with a position never being reduced before, we can reduce l_p positions, where $l_p = \left\lfloor \frac{b}{\log_2(q)} \right\rfloor$. Otherwise, when the first position is partially reduced in the previous step and we need β' categories to further reduce this position, we can in total fully reduce l_p positions, where $l_p = 1 + \left\lfloor \frac{b - \log_2(\beta')}{\log_2(q)} \right\rfloor$.

For this smooth-plain BKW step, we compute

$$C_{pbkw} \mathrel{+}= ((n + 1 - l_{red}) \cdot m + C_{d,pbkw}),$$

where $C_{d,pbkw} = m$ is the cost of modulus switching for the last partially reduced position in this step. We then update the number of the reduced positions, $l_{red} \mathrel{+}= l_p$.

After iterating for t_1 times, we could compute C_{pbkw} and l_{red}. We will continue updating l_{red} and denote n_{pbkw} the length reduced by the smooth-plain BKW steps.

Smooth-LMS steps before the multiplication of 2. We assume that the final noise contribution from each position reduced by LMS is similar, bounded by a preset value σ_{set}. Since the noise variable generated in the i-th ($0 \leq i \leq t_2 - 1$) Smooth-LMS step will be added by $2^{t_2+t_3-i}$ times and also be multiplied by 2, we compute $\sigma_{set}^2 = \frac{2^{t_2+t_3-i} \times 4 C_{i,LMS1}^2}{12}$, where $C_{i,LMS1}$ is the length of the interval after the LMS reduction in this step. We use $\beta_{i,LMS1}$ categories for one position, where $\beta_{i,LMS1} = \lceil \frac{q}{C_{i,LMS1}} \rceil$. Similar to smooth-plain BKW steps, if this step starts with an new position, we can reduce l_p positions, where $l_p = \lfloor \frac{b}{\log_2(\beta_{i,LMS1})} \rfloor$. Otherwise, when the first position is partially reduced in the previous step and we need $\beta'_{p,i,LMS1}$ categories to further reduce this position, we can in total fully reduce l_p positions, where $l_p = 1 + \lfloor \frac{b - \log_2(\beta'_{p,i,LMS1})}{\log_2(\beta_{i,LMS1})} \rfloor$. Let C_{LMS1} be the cost of Smooth-LMS steps before the multiplication of 2, which is innitialized to 0. For this step, we compute

$$C_{LMS1} \mathrel{+}= (n + 1 - l_{red}) \cdot m,$$

and then update the number of the reduced positions, $l_{red} \mathrel{+}= l_p$.

After iterating t_2 times, we compute C_{LMS1} and l_{red}. We expect $l_{red} = n - n_t$ ($n_t \leq n_{limit}$) positions have been fully reduced and will continue updating l_{red}.

Smooth-LMS steps after the multiplication of 2. The formulas are similar to those for Smooth-LMS steps before the multiplication of 2. The difference is

[6] The number of categories is doubled compared with the LF2 setting for LPN. The difference is that we could add and subtract samples for LWE.

σ_f	q	$D(\mathcal{X}_{\sigma_f,2q}\|\|U_{2q})$	$\mathbb{E}_{z=t}[D(e_{z=t}\|\|U_b)]$
$0.5q$	1601	-2.974149	-2.974995
$0.6q$	1601	-4.577082	-4.577116
$0.7q$	1601	-6.442575	-6.442576
$0.8q$	1601	-8.582783	-8.582783

Table 1. The comparison between $D(\mathcal{X}_{\sigma_f,2q}||U_{2q})$ and $\mathbb{E}_{z=t}[D(e_{z=t}||U_b)]$

that the noise term is no longer multiplied by 2, so we have $\sigma_{set}^2 = \frac{2^{t_3-i}C_{i,LMS2}^2}{12}$, for $0 \le i \le t_3 - 1$. Also, we need to track the a vector of length n_t for the later distinguisher. The cost is

$$C_{LMS2} = t_3 \cdot (n_t + 1) \cdot m.$$

We also need to count the cost for multiplying samples by 2 and the mod2 operations, and the LMS decoding cost, which are

$$C_{mulMod} = 2 \cdot (n_t + 1) \cdot m,$$
$$C_{dec} = (n - n_{pbkw} + t_2 + t_3) \cdot m.$$

FWHT distinguisher and partial guessing. After the LWE-to-LPN transformation, we have an LPN problem with dimension n_t and m instance. We perform partial guessing on n_{guess} positions, and use FWHT to recover the remaining $n_{FWHT} = n_t - n_{guess}$ positions. The cost is,

$$C_{distin} = 2^{n_{guess}} \cdot ((n_{guess} + 1) \cdot m + n_{FWHT} \cdot 2^{n_{FWHT}}).$$

6.3 The Data Complexity

We now discuss the data complexity of the new FWHT distinguisher. In the integer form, we have the following equation,

$$z_j = \sum_{i=0}^{n_t-1} s_i a_{ij} + 2E_j + k_j \cdot q.$$

If $|\sum s_i a_{ij} + 2E_j| < q/2$ then $k_j = 0$. Then the equation can be reduced mod 2 without error. In general, the error is $e_j = k_j \bmod 2$.

We employ a smart FWHT distinguisher with soft received information, as described in Section 5. From [29], we know the sample complexity can be approximated as $m \approx \frac{4 \ln n_t}{\mathbb{E}_{z=t}[D(e_{z=t}||U_b)]}$.

For different value of z_j, the distribution of e_j is different. The maximum bias is achieved when $z_j = 0$. In this sense, we could compute the divergence as

$$\mathbb{E}_{z=t}[D(e_{z=t}||U_b)] = \sum_{t \in \mathbb{Z}_q} \Pr[z = t]\, D(e_{z=t}||U_b)$$
$$= \sum_{t \in \mathbb{Z}_q} \Pr[z = t]\, (\sum_{i=0}^{1} \Pr[e_{z=t} = i] \log(2 \cdot \Pr[e_{z=t} = i]))$$

where e_z is the Bernoulli variable conditioned on the value of z and U_b the uniform distribution over the binary field.

Following the previous research [4], we approximate the noise $\sum s_i a_{ij} + 2E_j$ as discrete Gaussian with standard deviation σ_f. If σ_f is large, the probability $\Pr[z = t]$ is very close to $1/q$. Then, the expectation $\mathbb{E}_{z=t, t \in \mathbb{Z}_q}[D(e_{z=t} || U_b)]$ can be approximated as

$$\sum_{t \in \mathbb{Z}_q} \sum_{i=0}^{1} \Pr[z = t] \Pr[e_{z=t} = i] \log(2q \cdot \Pr[e_{z=t} = i, z = t]),$$

i.e., the divergence between a discrete Gaussian with the same standard deviation and a uniform distribution over $2q$, $D(\mathcal{X}_{\sigma_f, 2q} || U_{2q})$. We numerically computed that the approximation is rather accurate when the noise is sufficiently large (see Table 1). In conclusion, we use the formula

$$m \approx \frac{4 \ln n_t}{D(\mathcal{X}_{\sigma_f, 2q} || U_{2q})},$$

to estimate the data complexity of the new distinguisher. It remains to control the overall variance σ_f^2. Since we assume that the noise contribution from each reduced position by LMS is the same and the multiplication of 2 will double the standard deviation, we can derive $\sigma_f^2 = 4 * 2^{t_1+t_2+t_3}\sigma^2 + \sigma^2\sigma_{set}^2(n - n_{pbkw})$.

Note: The final noise is a combination of three parts, the noise from the LWE problem, the LMS steps before the multiplication by 2, and the LMS steps after the multiplication by 2. The final partial key recovery problem is equivalent to distinguishing a discrete Gaussian from uniform with the alphabet size doubled. We see that with the multiplication by 2, the variances of the first and the second noise parts are increased by a factor of 4, but the last noise part does not expand. This intuitively explains the gain of the new binary distinguisher.

6.4 In Summary

We have the following theorem to estimate the complexity of the attack.

Theorem 1. *The time complexity of the new algorithm is*

$$C = C_{pbkw} + C_{LMS1} + C_{LMS2} + C_{dec} + C_{distin} + C_{mulMod},$$

under the condition that

$$m \geq \frac{4 \ln n_t}{D(\mathcal{X}_{\sigma_f, 2q} || U_{2q})},$$

where $\sigma_f^2 = 4 * 2^{t_1+t_2+t_3}\sigma^2 + \sigma^2\sigma_{set}^2(n - n_{pbkw})$.

n	q	α	BKW-FWHT-SR	Coded-BKW	usvp ENU	usvp Sieve	dec ENU	dec Sieve	dual ENU	dual Sieve
				LWE estimator [7]						
40	1601	0.005	34.4	42.6	**31.4**	41.5	34.7	44.6	39.1	47.5
		0.010	39.3	43.7	**34.0**	44.8	36.3	44.9	51.1	57.9
		0.015	**42.4**	52.6	42.5	54.2	43.1	50.6	61.5	64.4
		0.020	**46.2**	52.6	∞	∞	51.9	58.2	73.1	75.9
		0.025	**48.3**	52.7	∞	∞	59.2	66.1	84.7	85.4
		0.030	**50.0**	52.7	∞	∞	67.1	68.9	96.3	92.5
45	2027	0.005	37.7	55.2	**31.8**	41.9	35.0	44.8	41.5	51.6
		0.010	43.5	55.2	**39.5**	51.2	41.2	48.2	57.0	64.6
		0.015	**48.3**	55.2	50.4	61.3	51.2	58.3	74.3	74.9
		0.020	**51.2**	55.2	∞	∞	61.1	65.0	86.8	86.1
		0.025	**54.1**	55.3	∞	∞	71.0	71.4	100.7	95.0
		0.030	**56.3**	64.1	∞	∞	80.2	78.7	116.2	104.1
50	2503	0.005	41.8	46.4	**32.4**	42.6	35.5	45.1	46.7	58.0
		0.010	48.7	56.0	**46.0**	57.5	47.6	54.1	66.8	65.4
		0.015	**52.5**	56.8	∞	∞	60.8	63.6	84.9	83.5
		0.020	**56.4**	61.9	∞	∞	72.1	72.1	101.9	96.5
		0.025	**59.3**	66.1	∞	∞	83.5	80.8	120.0	105.7
		0.030	**63.3**	66.3	∞	∞	94.2	89.1	134.0	115.6
70	4903	0.005	58.3	62.3	**52.3**	54.2	55.2	63.3	76.2	75.9
		0.010	**67.1**	73.7	∞	∞	80.4	77.1	111.3	98.9
		0.015	**73.3**	75.6	∞	∞	102.5	93.2	146.0	118.0
120	14401	0.005	100.1	110.5	133.0	**93.2**	135.5	111.4	181.9	133.2
		0.010	**115.1**	124.0	∞	∞	195.0	150.4	266.2	165.7
		0.015	**127.0**	136.8	∞	∞	246.4	183.2	334.0	209.8

Table 2. Estimated time complexity comparison (in $\log_2(\cdot)$) for solving LWE instances in the TU Darmstadt LWE challenge [2]. Here unlimited number of samples are assumed. The last columns show the complexity estimation from the LWE estimator [7]. "ENU" represents the enumeration cost model is employed and "Sieve" represents the sieving cost model is used. Bold-faced numbers are the smallest among the estimations with these different approaches.

6.5 Numerical Estimation

We numerically estimate the complexity of the new algorithm BKW-FWHT-SR (shown in Table 2). It improves the known approaches when the noise rate (represented by α) becomes larger. We should note that compared with the previous BKW-type algorithms, the implementation is much easier though the complexity gain might be mild.

7 A New BKW Algorithm Implementation for Large LWE Problem Instances

We have a new implementation of the BKW algorithm that is able to handle very large LWE problem instances. The code is written in C, and much care has been taken to be able to handle instances with a large number of samples.

A key success factor in the software design was to avoid unnecessary reliance on RAM, so we have employed file-based storage where necessary and practically possible. The implementation includes most known BKW reduction step, FFT and FWHT-based guessing methods, and hybrid guessing approaches.

For our experiments, presented in Section 8, we assembled a machine with an ASUS PRIME X399-A motherboard, a 4.0GHz Ryzen Threadripper 1950X processor and 128GiB of 2666MHz DDR4 RAM. While the machine was built from standard parts with a limited budget, we have primarily attempted to maximize the amount of RAM and the size and read/write speeds of the fast SSDs for overall ability to solve large LWE problem instances. The implementation is publicly available at https://github.com/FBBL/fbbl.

We describe below how we dealt with some interesting performance issues.

File-based Sample Storage The implementation does not assume that all samples can be stored in RAM, so instead they are stored on file in a special way. Samples are stored sorted into their respective categories. For simplicity, we have opted for a fixed maximum number of samples per category. The categories are stored sequentially on file, each containing its respective samples (possibly leaving some space if the categories are not full). A category mapping, unique for each reduction type, defines what category index a given sample belongs to[7].

Optional Sample Amplification We support optional sample amplification. That is, if a problem instance has a limited number of initial samples (e.g., the Darmstadt LWE challenge), then it is possible to combine several of these to produce new samples (more, but with higher noise).

While this is very straightforward in theory, we have noticed considerable performance effects when this recombination is performed naïvely. For example, combining triplets of initial samples using a nested loop is problematic in practice for some instances, since some initial samples become over-represented – Some samples are used more often than others when implemented this way.

We have solved this by using a Linear Feedback Shift Register to efficiently and pseudo-randomly distribute the selection of initial samples more evenly.

Employing Meta-Categories For some LWE problem instances, using a very high number of categories with few samples in each is a good option. This can be

[7] In this section a category is defined slightly differently from the rest of the paper. A category together with its adjacent category are together what we simply refer to as a category in the rest of the paper.

problematic to handle in an implementation, but we have used meta-categories to handle this situation. For example, using plain BKW reduction steps with modulus q and three positions, we end up with q^3 different categories. With q large, an option is to use only two out of the three position values in a vector to first map it into one out of q^2 different meta-categories. When processing the (meta-)categories, one then needs an additional pre-processing in form of a sorting step in order to divide the samples into their respective (non-meta) categories (based on all three position values), before proceeding as per usual.

We have used this implementation trick to, for example, implement plain BKW reduction for three positions. One may think of the process as brute-forcing one out of three positions in the reduction step.

Secret Guessing with FFT and FWHT The same brute-forcing techniques are also useful for speeding up the guessing part of the solver. We have used this to improve the FFT and FWHT solvers in the corresponding way.

For the FWHT case, if the number of positions to guess is too large for the RAM to handle, we leave some of them to brute-force. This case differs from the above by the fact that binary positions are brute-forced (so more positions can be handled) and that the corresponding entries in the samples must be reduced.

8 Experimental Results

In this section we report the experimental results obtained in solving some LWE problems. Our main goal was to confirm our theory and to prove that BKW algorithms can be used in practice to solve relatively large instances. Therefore, there is still room to run a more optimized code (for example, we did not use any parallelization in our experiments) and to make more optimal parameter choices (we generally used more samples than required and no brute-force guessing techniques were used).

We considered two different scenarios. In the first case, we assumed for each LWE instance to have access to an arbitrary large number of samples. Here we create the desired amount of samples ourselves[8]. In the second case, we considered instances with a limited number of samples. An LWE problem is "solved" when the binary secret is correctly guessed, for the reasons explained in Section 5.3.

Unlimited Number of Samples We targeted the parameter choices of the TU Darmstadt challenges [2]. For each instance, we generated as many initial samples as needed according to our estimations. In Table 3 we report the details of the largest solved instances. Moreover, in Example 2 we present our parameter choices for one of these.

[8] we used rounded Gaussian noise for simplicity of implementation.

n	q	α	number of samples	running time
40	1601	0.005	45 million	12 minutes
40	1601	0.01	1.6 billion	12 hours
45	2027	0.005	1.1 billion	13 hours

Table 3. Experimental results on target parameters.

Example 2. Let us consider an LWE instance with $n = 40$, $q = 1601$ and $\sigma = 0.005 \cdot q$. To successfully guess the secret, we first performed 8 smooth-plain BKW steps reducing 18 positions to zero. We used the following parameters.

$$n_i = 2, \quad C_i = 1, \qquad \text{for} \quad i = 1, \dots, 8,$$
$$(C'_1, C'_2, C'_3, C'_4, C'_5, C'_6, C'_7, C'_8) = (165, 30, 6, 1, 165, 30, 6, 1).$$

Note that $C'_4 = C'_8 = 1$. In this way, we exploited the smoothness to zero 9 positions every 4 steps. For this reason, we start steps 5 and 9 by skipping one position. Finally, we did 5 smooth-LMS steps using the following parameters:

$$(n_9, n_{10}, n_{11}, n_{12}, n_{13}) = (3, 4, 4, 5, 6)$$
$$(C_9, C_{10}, C_{11}, C_{12}, C_{13}) = (17, 24, 34, 46, 66)$$
$$(C'_9, C'_{10}, C'_{11}, C'_{12}) = (46, 66, 23, 81).$$

These parameters are chosen in such a way that the number of categories within each step is $\approx 13M$ and $C_i \approx \sqrt{2}C_{i-1}$. We used $\approx 40M$ samples in each step so that each category contained 3 samples in average. This way we are guaranteed to have enough samples in each step.

Limited Number of Samples As a proof-of-concept, we solved the original TU Darmstadt LWE challenge instance [2] with parameters $n = 40$, $\alpha = 0.005$ and the number of samples limited to $m = 1600$. We did this by sample amplifying with triples of samples, taking 7 steps of smooth-plain BKW on 17 entries, 5 steps of smooth-LMS on 22 entries and 1 position was left to brute-force. The overall running time was of 3 hours and 39 minutes.

9 Conclusions and Future Work

We introduced a novel and easy approach to implement the BKW reduction step which allows balancing the complexity among the iterations, and an FWHT-based guessing procedure able to correctly guess the secret with relatively large noise level. Together with a file-based approach of storing samples, the above define a new BKW algorithm specifically designed to solve practical LWE instances. We leave optimization of the implementation, including parallelization, for future work.

Acknowledgements

This work was supported in part by the Swedish Research Council (Grants No. 2015-04528 and 2019-04166), the Norwegian Research Council (Grant No. 247742/070), and the Swedish Foundation for Strategic Research (Grant No. RIT17-0005 and strategic mobility grant No. SM17-0062). This work was also partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

1. NIST Post-Quantum Cryptography Standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization, accessed: 2018-09-24
2. TU Darmstadt Learning with Errors Challenge. https://www.latticechallenge.org/lwe_challenge/challenge.php, accessed: 2020-05-01
3. Albrecht, M., Cid, C., Faugere, J.C., Fitzpatrick, R., Perret, L.: On the complexity of the arora-ge algorithm against lwe (2012)
4. Albrecht, M.R., Cid, C., Faugère, J., Fitzpatrick, R., Perret, L.: On the complexity of the BKW algorithm on LWE. Des. Codes Cryptogr. 74(2), 325–354 (2015)
5. Albrecht, M.R., Ducas, L., Herold, G., Kirshanova, E., Postlethwaite, E.W., Stevens, M.: The general sieve kernel and new records in lattice reduction. In: Rijmen, V., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part II. pp. 717–746. Lecture Notes in Computer Science, Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019)
6. Albrecht, M.R., Faugère, J.C., Fitzpatrick, R., Perret, L.: Lazy modulus switching for the BKW algorithm on LWE. In: Krawczyk, H. (ed.) PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, vol. 8383, pp. 429–445. Springer, Heidelberg, Germany, Buenos Aires, Argentina (Mar 26–28, 2014)
7. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Mathematical Cryptology 9(3), 169–203 (2015)
8. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) Advances in Cryptology – CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 595–618. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2009)
9. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part I. Lecture Notes in Computer Science, vol. 6755, pp. 403–415. Springer, Heidelberg, Germany, Zurich, Switzerland (Jul 4–8, 2011)
10. Baignères, T., Junod, P., Vaudenay, S.: How far can we go beyond linear cryptanalysis? In: Lee, P.J. (ed.) Advances in Cryptology – ASIACRYPT 2004. Lecture Notes in Computer Science, vol. 3329, pp. 432–450. Springer, Heidelberg, Germany, Jeju Island, Korea (Dec 5–9, 2004)
11. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: Krauthgamer, R. (ed.) 27th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 10–24. ACM-SIAM, Arlington, VA, USA (Jan 10–12, 2016)

12. Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Stinson, D.R. (ed.) Advances in Cryptology – CRYPTO'93. Lecture Notes in Computer Science, vol. 773, pp. 278–291. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 22–26, 1994)
13. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. In: 32nd Annual ACM Symposium on Theory of Computing. pp. 435–440. ACM Press, Portland, OR, USA (May 21–23, 2000)
14. Chose, P., Joux, A., Mitton, M.: Fast correlation attacks: An algorithmic point of view. In: Knudsen, L.R. (ed.) Advances in Cryptology — EUROCRYPT 2002. pp. 209–221. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
15. Delaplace, C., Esser, A., May, A.: Improved low-memory subset sum and LPN algorithms via multiple collisions. In: 17th IMA International Conference on Cryptography and Coding. pp. 178–199. Lecture Notes in Computer Science, Springer, Heidelberg, Germany, Oxford, UK (Dec 2019)
16. Duc, A., Tramèr, F., Vaudenay, S.: Better algorithms for LWE and LWR. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 173–202. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015)
17. Esser, A., Heuer, F., Kübler, R., May, A., Sohler, C.: Dissection-BKW. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 638–666. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)
18. Esser, A., Kübler, R., May, A.: LPN decoded. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 486–514. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017)
19. Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, Part I. Lecture Notes in Computer Science, vol. 8873, pp. 1–20. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014)
20. Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. Journal of Cryptology 33(1), 1–33 (Jan 2020)
21. Guo, Q., Johansson, T., Mårtensson, E., Stankovski, P.: Coded-BKW with sieving. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 323–346. Springer, Heidelberg, Germany, Hong Kong, China (Dec 3–7, 2017)
22. Guo, Q., Johansson, T., Mårtensson, E., Stankovski Wagner, P.: On the asymptotics of solving the LWE problem using coded-bkw with sieving. IEEE Trans. Inf. Theory 65(8), 5243–5259 (2019)
23. Guo, Q., Johansson, T., Stankovski, P.: Coded-BKW: Solving LWE using lattice codes. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 23–42. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)
24. Herold, G., Kirshanova, E., May, A.: On the asymptotic complexity of solving LWE. Designs, Codes and Cryptography 86(1), 55–83 (2018)
25. Kirchner, P.: Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377 (2011), http://eprint.iacr.org/2011/377
26. Kirchner, P., Fouque, P.A.: An improved BKW algorithm for LWE with applications to cryptography and lattices. In: Gennaro, R., Robshaw, M.J.B. (eds.)

Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 43–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)
27. Levieil, É., Fouque, P.A.: An improved LPN algorithm. In: Prisco, R.D., Yung, M. (eds.) SCN 06: 5th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 4116, pp. 348–359. Springer, Heidelberg, Germany, Maiori, Italy (Sep 6–8, 2006)
28. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) Topics in Cryptology – CT-RSA 2011. Lecture Notes in Computer Science, vol. 6558, pp. 319–339. Springer, Heidelberg, Germany, San Francisco, CA, USA (Feb 14–18, 2011)
29. Lu, Y., Meier, W., Vaudenay, S.: The conditional correlation attack: A practical attack on Bluetooth encryption. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 97–117. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005)
30. Mårtensson, E.: The asymptotic complexity of coded-bkw with sieving using increasing reduction factors. In: 2019 IEEE International Symposium on Information Theory (ISIT). pp. 2579–2583 (2019)
31. Meier, W., Staffelbach, O.: Fast correlation attacks on certain stream ciphers. Journal of Cryptology 1(3), 159–176 (Oct 1989)
32. Mulder, E.D., Hutter, M., Marson, M.E., Pearson, P.: Using bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA: extended version. J. Cryptographic Engineering 4(1), 33–45 (2014)
33. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing. pp. 84–93. ACM Press, Baltimore, MA, USA (May 22–24, 2005)
34. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science. pp. 124–134. IEEE Computer Society Press, Santa Fe, New Mexico (Nov 20–22, 1994)