Memory-Efficient BKW Algorithm for Solving the LWE Problem

Yu Wei[1,2], Lei Bi[1,2](✉), Xianhui Lu[1,2], and Kunpeng Wang[1,2]

[1] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{weiyu,bilei,luxianhui,wangkunpeng}@iie.ac.cn

Abstract. The study of attack algorithms for the Learning with Errors (LWE) problem is crucial for the cryptanalysis of LWE-based cryptosystems. The BKW algorithm has gained significant attention as an important combinatorial attack for solving LWE. However, its exponential time and memory requirements severely limit its practical applications, even with medium-sized parameters. In this paper, we present a memory-efficient BKW algorithm for LWE, which extends Bogos's work [Asiacrypt'16] on the Learning Parity with Noise (LPN) problem. While their work improved efficiency, it overlooked the high memory demands of the BKW algorithm. We address this with two key improvements. First, we propose an efficient reduction technique for low-memory regimes, c-sum-PCS-reduce, which combines the c-sum technique with Parallel Collision Search (PCS) to achieve a better time-memory trade-off. Second, we present an improved memory-optimized finite automaton for our optimized BKW algorithm by incorporating several efficient memory-saving reduction techniques and pruning potential high-memory paths. Our algorithm, using graphs as a meta tool, can automatically identify the optimal reduction path within the graph, aiming to reduce both time and memory complexities. Compared to the state-of-the-art coded-BKW in the lattice-estimator, our algorithm achieves time complexity improvements ranging from $2^{3.3}$ to $2^{26.2}$. Furthermore, memory complexity is improved, with reductions ranging from $2^{9.7}$ to $2^{71.3}$.

Keywords: Post-quantum cryptanalysis · Learning with errors problem · The BKW algorithm · Time-memory trade-off

1 Introduction

The Learning with Errors (LWE) problem [37] is a highly promising problem in lattice-based post-quantum cryptography. Specifically, the LWE problem can be regarded as a set of linear equations with noise over $\mathbb{Z}_q$. An LWE oracle outputs uniformly distributed vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$ and $\langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \mod q$, where $\mathbf{s} \in \mathbb{Z}_q^n$ is the secret vector and e_i is a noise typically sampled from a discrete Gaussian distribution with zero mean and variance σ^2. As many post-quantum public-key schemes base their security on the LWE problem, designing efficient algorithms to solve LWE and assessing its hardness are crucial for reliably determining strong parameters of LWE-based post-quantum cryptography.

The attack algorithms consist of three main categories: lattice attacks, algebraic attacks, and combinatorial attacks. Lattice attacks, such as primal attack [42] and dual attack [25], generally reduce the LWE problem to a lattice problem, which is then solved using lattice reduction algorithms like LLL and BKZ [11,30,32,33,38]. Algebraic attacks [6] typically transform the LWE problem into a system of algebraic equations with unknown variables and use algebraic techniques, such as Gröbner bases, to solve it. Combinatorial attacks generally include the BKW algorithms [2,3,21,23] and the Meet-in-the-Middle attack [15].

In this paper, we primarily focus on the BKW algorithm, which is a significant combinatorial algorithm in LWE-based cryptanalysis. Generally, it has two phases: the reduction phase progressively reduces the dimension of LWE, and the solving phase recovers the remaining entries of the secret. When solving an LWE problem with dimension n, where $n = a \cdot b$ and $a, b \in \mathbb{N}$, the n-dimensional secret $\mathbf{s}$ is divided into a blocks, each containing b entries. The reduction phase is a block-wise variant of Gaussian elimination, which gradually reduces the problem size by b entries at each step, resulting in a smaller LWE problem with increased noise. The solving phase then recovers the left b entries of the secret $\mathbf{s}$ at a time. After recovering a portion of $\mathbf{s}$, the samples are refreshed, and the process iterates until all remaining entries are recovered.

In previous work, the BKW algorithm was initially applied to the learning parity with noise (LPN) problem [7,29] and later adapted by Albrecht et al. [2] to LWE. The first BKW algorithm for solving LWE [2] required the construction of numerous tables, resulting in significant time and memory consumption. To improve the reduction phase, subsequent work incorporated lazy modulus switching (LMS) [3], which was particularly effective for relatively small secrets. Additionally, both approaches [2, 3] relied on exhaustive search in the solving phase, leading to high time complexity. In 2015, Kirchner et al. [27] introduced a quantization step that generalizes and fine-tunes modulus switching, thereby optimizing the BKW algorithm. In 2015, Duc et al. [16] improved the solving phase by applying the Fast Fourier Transform (FFT). Subsequently, Guo et al. [23] enhanced the BKW algorithm by incorporating linear lattice codes and lattice sieving [21–23], resulting in the development of the coded-BKW algorithm, which is the most efficient and widely used BKW algorithm. Recent years have also seen improvements in analyzing sample complexity and noise distributions [24, 36].

We have identified several unresolved issues in previous research on the BKW algorithm for solving LWE. Many BKW algorithms suffer from high memory and time complexities, which makes them less competitive than lattice attacks [25, 42]. Moreover, potential improvements of various reduction and solving techniques under memory constraints remain largely unexplored. Although recent studies on BKW algorithm for solving LPN [17, 31] have explored time-memory trade-offs, a thorough analysis of such trade-offs within the context of LWE-solving BKW algorithms is still lacking.

This paper introduces a memory-efficient BKW algorithm designed to solve the LWE problem, building on prior works applying BKW algorithms to LPN [8, 9, 12, 20, 29, 31, 41]. We extend Bogos's work [9] on LPN to design an LWE-solving BKW algorithm, using graphs as a meta tool. However, their research primarily focused on improving efficiency, overlooking BKW's high memory consumption. We address this by introducing several efficient reduction techniques for low-memory regimes and an improved memory-optimized finite automaton. This leads to a more efficient BKW algorithm with reduced time and memory complexities when solving LWE.

1.1 Contributions

This paper makes the following major contributions. The source code is provided at https://github.com/hfafsjlq/BKW.git.

Unified Framework for Reduction and Solving Techniques. We present a unified framework encompassing all existing reduction and solving techniques of the BKW algorithm for LWE, providing a consistent analysis of the time complexity for each method. This unified framework serves as the basis for designing our optimized BKW algorithm. Furthermore, we correct several inaccuracies in previous complexity analyses, providing more accurate estimations.

Optimized BKW Algorithm for LWE. We propose an optimized BKW algorithm to solve the LWE problem, utilizing graphs as a meta tool, inspired by the work of Bogos [9] on the LPN problem. The construction of the graph $G = (V, E)$ for this algorithm is based on the transition rules of a finite automaton, specifically adapted to the LWE problem as an extension of Bogos's work [9]. In our algorithm, various reduction techniques are formalized as edges in the graph, while the states of the LWE samples—including vector dimensions and the number of samples—are formalized as vertices. Consequently, the reduction phase of the BKW algorithm is represented as a path in the graph G, where each edge corresponds to a reduction technique that progressively reduces the dimension of the LWE problem. For different LWE instances, we automate the search for the optimal reduction path within this graph, identifying the most efficient combination of reduction techniques to minimize the overall time complexity.

Memory-Efficient BKW Algorithm for LWE. The main drawback of the BKW algorithm for solving LWE (or LPN) is its huge memory consumption, which makes it impractical even for medium-sized parameters. We found that the natural extension of Bogos's work [9] on LPN to LWE fails to address this critical memory issue. Specifically, we identified two key limitations in the natural extension: a limited variety of reduction techniques and a focus solely on efficiency without considering memory consumption. For instance, as shown in Fig. 2(a), all reduction techniques rely on traditional pairwise reduction (i.e., 2-sum) and do not incorporate modern memory-saving methods, such as c-sum reduction for $c \geq 2$. Moreover, the repeated application of the *Discard-reduce* technique can lead to significant memory overhead. To address these issues, we refined the natural extension of their framework and proposed a memory-efficient BKW algorithm specifically designed for LWE. Our approach introduces two key improvements:

- **Reduction Techniques for Low-Memory Regimes.** We introduce several efficient reduction techniques for our BKW algorithm under memory constraints. These techniques are based on the c-sum method for $c \geq 2$, which searches for c-tuples of vectors that sum to zero at selected b positions. Besides the existing c-sum techniques, such as c-sum-naive-reduce [17] and c-sum-dissect-reduce [17], which can be directly applied to LWE-solving BKW, we introduce a new reduction technique, c-sum-PCS-reduce, inspired by Delaplace's work [12]. This technique combines c-sum with Parallel Collision Search (PCS) to improve efficiency under memory constraints. Specifically, it symmetrically splits a c-sum problem into two $\frac{c}{2}$-sum problems and utilizes PCS to accelerate collision search.
  We present the time and memory complexities of the corresponding c-sum-BKW algorithms in Table 1, along with a comparison of their time-memory trade-offs for LWE in Table 2 and Fig. 1. As illustrated in Fig. 1, when memory is constrained to $2^{0.3n}$, c-sum-PCS-BKW achieves lower time complexity than c-sum-dissect-BKW [17]. Furthermore, when memory is limited to less than $2^{0.63n}$, c-sum-PCS-BKW shows significant advantages in both time and memory complexities over c-sum-naive-BKW [17].

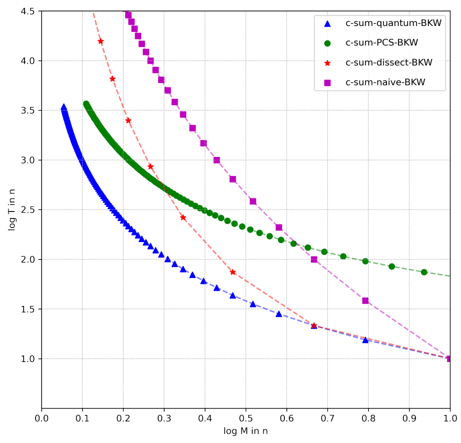

Fig. 1: Comparison of the time-memory trade-offs for various c-sum-BKW algorithms utilizing different c-sum reduction techniques (both classical and quantum).

- **Improved Memory-Optimized Finite Automaton.** We propose a memory-optimized finite automaton tailored for our BKW algorithm in solving LWE problems. First, we incorporate several efficient c-sum reduction techniques into the finite automaton to balance time and memory consumption, as shown in Fig. 2(b). Second, we prune high-memory paths and adjust the order of state transitions in the automaton. We show that, in memory-constrained scenarios, no techniques from *Reduce-techniques* $\rightarrow$ *Discard-reduce* can occur in the optimal reduction path of the BKW algorithm. Thus, the naturally extended automaton can be significantly simplified by eliminating many impossible paths. We also adjust the order of *Discard-reduce*, placing it as the first step. On-the-fly generation allows samples to be generated and discarded instantly if a non-zero value is detected. A detailed comparison between the trivially extended automaton and our memory-optimized automaton is shown in Fig. 2. This improved automaton enables the reconstruction of the graph for our BKW algorithm, effectively solving various LWE instances in low-memory regimes.

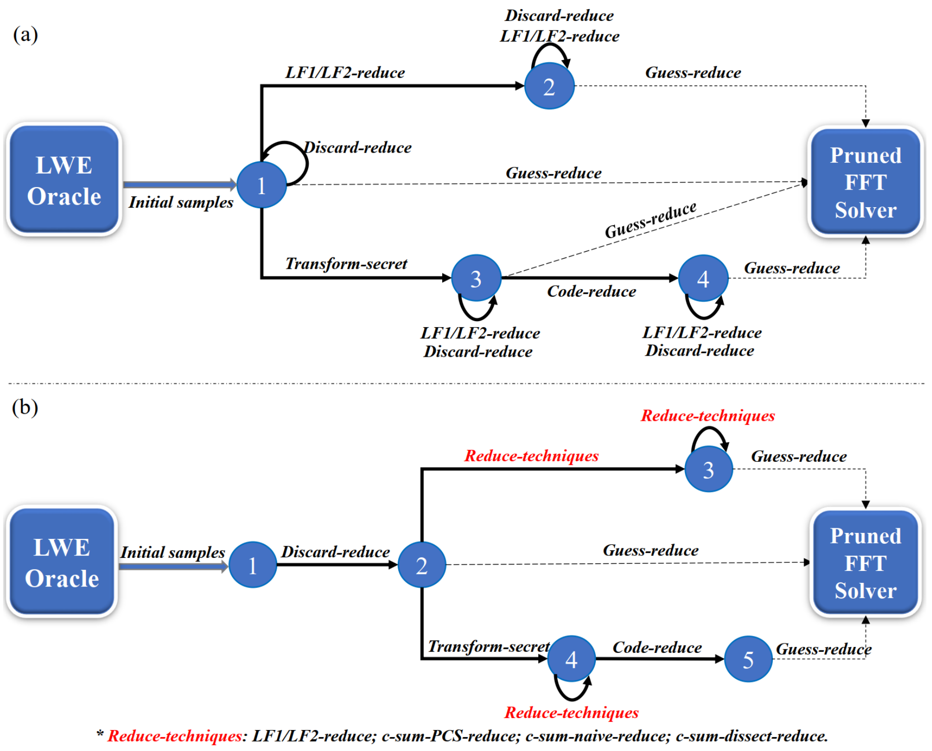

Fig. 2: A detailed comparison between the trivially extended automaton and our memory-optimized automaton. (a) Finite automaton for our optimized BKW algorithm for solving LWE: a natural extension of Bogos's work on LPN [9]. (b) Improved memory-optimized automaton for our optimized BKW algorithm for solving LWE.

Complexity Estimation and Comparison. We estimate the time and memory complexities of our BKW algorithm and compare them with the state-of-the-art coded-BKW algorithm [23] from the lattice-estimator [4], applied to LWE instances from Regev [37], Lindner-Peikert [30], and the TU Darmstadt challenge [1]. As shown in Table 3, our improved BKW algorithm achieves time complexity improvements ranging from 3.3 to 26.2 bits across all considered LWE instances. Furthermore, Table 4 demonstrates significant memory improvements compared to coded-BKW, particularly under a memory constraint of $2^{0.5n}$, where 16 out of 27 LWE instances consume memory less than $2^{0.5n}$. Overall, for all parameters tested [1,30,37], memory consumption is improved by factors ranging from $2^{9.7}$ to $2^{71.3}$.

1.2 Organization

The structure of this paper is as follows. We introduce the necessary preliminaries of the LWE problem and the BKW algorithm in Sect. 2. We systematically describe all existing reduction and solving techniques within a unified framework in Sect. 3. Sect. 4 extends Bogos's work [9] on LPN to develop a novel BKW algorithm for solving the LWE problem. Sect. 5 focuses on optimizing this algorithm to reduce both time and memory complexities. In Sect. 5.1, we present a reduction technique called c-sum-PCS-reduce, detailing its time-memory trade-off. Sect. 5.2 introduces improvements to the memory-optimized finite automaton, while Sect. 5.3 outlines search algorithms for finding the optimal reduction path in the graph. We employ our algorithm to solve various LWE instances in Sect. 6, presenting time and memory complexities along with comparisons. Finally, we conclude this paper in Sect. 7.

2 Preliminaries

2.1 Notations

Row vectors use bold lowercase (e.g., $\mathbf{s}$), and matrices use bold uppercase (e.g., $\mathbf{A}$). The L_2-norm of a vector $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ is given by $||\mathbf{x}|| = \sqrt{x_1^2 + \cdots + x_n^2}$. The Euclidean distance between two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$ is $||\mathbf{x} - \mathbf{y}||$, and their dot product is $\langle \mathbf{x}, \mathbf{y} \rangle$. Elements in $\mathbb{Z}_q$ are integers within the range $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$. The base-2 logarithm is denoted as $\log(\cdot)$. Drawing an element x uniformly from a domain D is expressed as $x \xleftarrow{U} D$.

2.2 The LWE Problem

Definition 1 (LWE$_{\mathbf{s},\chi_\sigma}$ distribution [37]). *Given a positive integer n, prime q, and a discrete Gaussian distribution χ_σ over $\mathbb{Z}_q$ with mean 0 and variance σ^2. Given a secret vector $\mathbf{s} \xleftarrow{U} \mathbb{Z}_q^n$, the LWE$_{\mathbf{s},\chi_\sigma}$ distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is obtained by selecting $\mathbf{a} \xleftarrow{U} \mathbb{Z}_q^n$, $e \leftarrow \chi_\sigma$, and returning*

$$(\mathbf{a}, z) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e \mod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q. \tag{1}$$

Search-LWE involves recovering the secret vector $\mathbf{s}$ from m samples of $\mathrm{LWE}_{\mathbf{s},\chi_\sigma}$, while decision-LWE involves distinguishing between samples drawn from the $\mathrm{LWE}_{\mathbf{s},\chi_\sigma}$ distribution and those from a uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

As discussed in many references [16,23], the properties of continuous Gaussian distributions can also be applied to discrete ones. If two independent random variables X and Y follow distributions χ_{σ_1} and χ_{σ_2}, respectively, their sum $X+Y$ can be regarded as following distribution $\chi_{\sqrt{\sigma_1^2+\sigma_2^2}}$.

The LWE problem can be formulated in matrix form. Select m samples $(\mathbf{a}_1, z_1), (\mathbf{a}_2, z_2), \cdots, (\mathbf{a}_m, z_m)$ from distribution $L_{\mathbf{s},\chi_\sigma}$, where $\mathbf{a}_i \in \mathbb{Z}_q^n$, $z_i \in \mathbb{Z}_q$. Let $\mathbf{y} = (y_1, y_2, \cdots, y_m) = \mathbf{sA}$ and $\mathbf{z} = (z_1, z_2, \cdots, z_m)$. Then we have

$$\mathbf{z} = \mathbf{sA} + \mathbf{e} \mod q, \tag{2}$$

where $\mathbf{A} = \left[\mathbf{a}_1^{\mathrm{T}} \mathbf{a}_2^{\mathrm{T}} \cdots \mathbf{a}_m^{\mathrm{T}}\right]$, $z_i = y_i + e_i = \langle \mathbf{a}_i, \mathbf{s}\rangle + e_i \mod q$ and e_i are drawn from χ_σ. Now the search LWE problem can be seen as a decoding problem. The matrix $\mathbf{A}$ represents the generator matrix for linear codes over $\mathbb{Z}_q$ and $\mathbf{z}$ represents the received message. The goal is to recover $\mathbf{s}$ by searching for the codeword $\mathbf{y} = \mathbf{sA}$ that minimizes the distance to $\mathbf{z}$.

2.3 The BKW Algorithm

We introduce the BKW algorithm for solving LWE, focusing on the reduction and solving phases.

The Reduction Phase. This phase is a variant of Gaussian elimination, aiming to reduce the dimension of the LWE problem. Given LWE samples written as $\mathbf{z} = \mathbf{sA} + \mathbf{e} \mod q$, where $\mathbf{z} \in \mathbb{Z}_q^m$, $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $n = a \cdot b$ and $a, b \in \mathbb{N}$. The BKW algorithm applies some reduction techniques to the columns of $\mathbf{A}$, progressively reducing some positions to zero.

We employ a sort-and-match strategy on $\mathbf{A}$, zeroing out a fixed number of b entries in each iteration. In the initial step, the columns of $\mathbf{A}$ are classified into distinct categories according to the values in their last b entries.

Elements with the same absolute value are classified into the same category. Then we search for collisions in each category and match them to create new sample vectors. If two vectors $\mathbf{a}_{i_1}^{\mathrm{T}}, \mathbf{a}_{i_2}^{\mathrm{T}}$ satisfy

$$\mathbf{a}_{i_1}^{\mathrm{T}} \pm \mathbf{a}_{i_2}^{\mathrm{T}} = (* \cdots * \underbrace{0 \cdots 0}_{b}), \tag{3}$$

a new vector $\hat{\mathbf{a}}^{\mathrm{T}} = \mathbf{a}_{i_1}^{\mathrm{T}} \pm \mathbf{a}_{i_2}^{\mathrm{T}}$ is obtained. The corresponding

$$z^{(1)} = z_{i_1} \pm z_{i_2} = \mathbf{s} \cdot \hat{\mathbf{a}}^{\mathrm{T}} + e^{(1)} \bmod q, \tag{4}$$

where $e^{(1)} = e_{i_1} \pm e_{i_2}$ is drawn from a discrete Gaussian distribution with a standard deviation of $\sqrt{2}\sigma$.

Repeating this step $(a-1)$ times zeros out the last $(a-1)\cdot b$ entries of vectors in matrix $\mathbf{A}$, leaving only the first b positions non-zero. The reduction phase of the BKW algorithm can be referenced in Fig. 3. As a result, we obtain a new LWE problem with a reduced dimension but increased noise, where the noise follows the discrete Gaussian distribution with a standard deviation of $\sqrt{2^{(a-1)}}\sigma$.

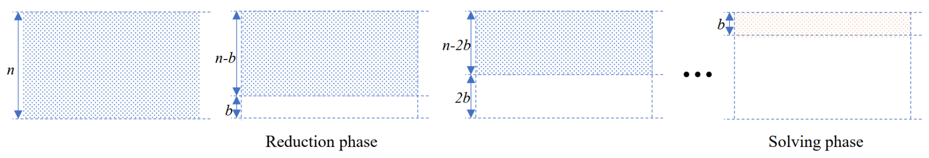

Fig. 3: Reduction and solving phases of the BKW algorithm. The blue rectangle represents the LWE sample matrix $\mathbf{A}$, which initially has n rows and is gradually reduced to b rows during the first stage.

The Solving Phase. This phase can recover the left b positions of the secret $\mathbf{s}$ at a time. Denote the LWE instances after the reduction of the form

$$z_i = \bar{\mathbf{s}} \cdot \bar{\mathbf{a}_i}^{\mathrm{T}} + \bar{e}_i \mod q, \quad i = 1, 2, \cdots, M \tag{5}$$

where $\bar{\mathbf{s}}$ and $\bar{\mathbf{a}}_i^{\mathrm{T}}$ are vectors of dimension b, the errors $\bar{e}_i$ follow a discrete Gaussian distribution $\chi_{\sqrt{2^{(a-1)}}\sigma}$. M is sample size required to successfully recover the secret $\bar{\mathbf{s}}$.

The simplest approach to recover the secret is to guess all possible values of $\bar{\mathbf{s}}$ and observe the resulting error distribution. For each candidate $\tilde{\mathbf{s}}$, we compute the corresponding error

$$\tilde{e}_i = z_i - \tilde{\mathbf{s}} \cdot \bar{\mathbf{a}}_i^{\mathrm{T}} \mod q. \tag{6}$$

A correct candidate $\tilde{\mathbf{s}}$ is guaranteed when $\tilde{e}_i$ is drawn from a discrete Gaussian distribution. Conversely, if $\tilde{e}_i$ follows a uniform distribution, the guess is wrong. The samples are refreshed upon recovering a portion of $\mathbf{s}$, and the process continues iteratively until all remaining entries are recovered.

3 BKW Techniques in a Unified Framework

This section introduces a unified framework that systematically describes all existing reduction and solving techniques, serving as the basis for designing our optimized BKW algorithm. We also correct several inaccuracies found in previous complexity estimations. For reduction techniques, we detail the noise variation, sample size changes, and time complexity. For solving techniques, we outline the success probability of secret recovery, the required number of samples, and time complexity.

3.1 Reduction Techniques

As described in Sect. 2.3, we assume that all reduction techniques begin with m samples, where the secret has dimension n and the standard deviations of the secret and noise are σ_s and σ, respectively. After one reduction step, we obtain m' samples (which may be more or fewer), with the secret's dimension reduced to $n' \leq n$ and the standard deviations updated to σ'_s and σ'.

We describe the existing reduction techniques as follows:

- **LF1-reduce** zeros out b rows of the sample matrix $\mathbf{A}$ in one reduction step [29]. It categorizes the column vectors of $\mathbf{A}$ based on the absolute values in the selected b positions. There are q^b possible values, and excluding the all-zero case, this results in $\frac{q^b-1}{2}$ categories. This method selects a fixed sample from each category and uses it to add or subtract other samples, generating new samples. This technique reduces the sample size by $\frac{q^b-1}{2}$ per step, with the standard deviation becoming $\sqrt{2}\sigma$.

  LF1-reduce(b): $n' = n - b;\ m' = m - \frac{q^b-1}{2};\ \sigma' = \sqrt{2}\sigma;\ \sigma'_s = \sigma_s$
  Complexity: $\mathcal{O}(n \cdot m)$

- **LF2-reduce** is a heuristic improvement of *LF1-reduce*, which adds or subtracts any sample pairs within each category to create updated samples [24]. This technique also reduces b dimensions per step, resulting in an increase in standard deviation to $\sqrt{2}\sigma$. It generates enough samples with a time complexity similar to *LF1-reduce* but increases the correlation between samples, thus requiring the independence heuristic discussed in several studies [8,16,29]. The expected number of new samples is given by $m' = \frac{m^2}{q^b-1} - \frac{m}{2}$. We have revised the estimate proposed by Bogos [8], $\frac{2m^2}{(q^b-1)^2} - \frac{m}{q^b-1}$, as their result omitted the factor $(q^b-1)/2$, which only considered samples within a single category, resulting in an inaccurate estimate.

  LF2-reduce(b): $n' = n - b;\ m' = \frac{m^2}{q^b-1} - \frac{m}{2};\ \sigma' = \sqrt{2}\sigma;\ \sigma'_s = \sigma_s$
  Complexity: $\mathcal{O}(n \cdot \max\{m, m'\})$

- **Transform secret** can convert the secret vector $\mathbf{s}$ from a uniform distribution to a noise distribution [5,28], which meets the requirements of certain reduction or solving techniques that aim for a minimal standard deviation of $\mathbf{s}$. Consider a LWE problem $\mathbf{z} = \mathbf{sA} + \mathbf{e} \mod q$, where $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$. By permuting the columns of $\mathbf{A}$, find the first n linearly independent columns, denoted by $\mathbf{A}_0$. Let $\mathbf{D} = \mathbf{A}_0^{-1}$, then $\hat{\mathbf{A}} = \mathbf{DA} = \left(\mathbf{I}, \hat{\mathbf{a}}_{n+1}^{\mathrm{T}}, \hat{\mathbf{a}}_{n+2}^{\mathrm{T}}, \cdots, \hat{\mathbf{a}}_{m}^{\mathrm{T}}\right)$. Calculate $\hat{\mathbf{z}} = \mathbf{z} - (z_1, z_2, \cdots, z_n)\,\hat{\mathbf{A}} = (\mathbf{0}, \hat{z}_{n+1}, \hat{z}_{n+2}, \cdots, \hat{z}_m)$, where the first n positions are zero. With the remaining $m-n$ samples, we derive a new LWE problem $\hat{\mathbf{z}} = \hat{\mathbf{s}}\hat{\mathbf{A}} + \mathbf{e} \mod q$, where the new secret $\hat{\mathbf{s}} = \mathbf{sD}^{-1} - (z_1, z_2 \cdots, z_n) = (-e_1, -e_2, \cdots, -e_n)$. Through this transformation, each entry of $\mathbf{s}$ follows the distribution χ_σ.

Transform secret: $n' = n$; $m' = m - n$; $\sigma' = \sigma$; $\sigma'_s = \sigma$
Complexity:$\mathcal{O}(n^2 + \frac{m' \cdot n^2}{\log_2 n - \log_2 \log_2 n})$

- **Code-reduce** is a reduction technique using a linear code $[n, n']$, first introduced by Guo et al. [20] for the LPN-solving BKW algorithm, and later extended to the LWE problem [23]. *Code-reduce* consists of two steps: decoding and reduction. Decoding maps sample vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$ to the nearest codeword $\mathbf{c}_i \in \mathbb{Z}_q^n$, while reduction allows sample vectors mapped to the same codeword to be reduced against each other.
Using an efficient decoding procedure, we represent $\mathbf{a}_i = \mathbf{c}_i + \mathbf{t}_i$, where $\mathbf{c}_i = \mathbf{c}'_i\mathbf{G}$, $\mathbf{c}'_i \in \mathbb{Z}_q^{n'}$, and $\mathbf{G} \in \mathbb{Z}_q^{n' \times n}$ is the generator matrix of the code. The vector $\mathbf{t}_i$ represents the decoding error introduced by this technique. The noisy LWE equations are given by:
$$\langle \mathbf{a}_i, \mathbf{s} \rangle + e_i = \langle \mathbf{c}'_i\mathbf{G}, \mathbf{s} \rangle + \langle \mathbf{t}_i, \mathbf{s} \rangle + e_i = \langle \mathbf{c}'_i, \mathbf{s}\mathbf{G}^{\mathrm{T}} \rangle + \langle \mathbf{t}_i, \mathbf{s} \rangle + e_i, i = 1, 2, ...m \quad (7)$$
where the new secret $\mathbf{s}' = \mathbf{s}\mathbf{G}^{\mathrm{T}} \in \mathbb{Z}_q^{n'}$ has dimension n', and the noise term becomes $e'_i = \langle \mathbf{t}_i, \mathbf{s} \rangle + e_i$.
Since the coding noise $\langle \mathbf{t}_i, \mathbf{s} \rangle$ depends on $\mathbf{s}$ and we aim to minimize it, applying *Transform secret* before *Code-reduce* is necessary. Multiple *Code-reduce* steps are not permitted, so variations in σ'_s are not considered here. Following previous work [20, 23], we assume that the decoding procedure has a linear time complexity and that the coding noise follows a Gaussian distribution with standard deviation σ_c.

Code-reduce: n'; $m' = m$; $\sigma' = \sqrt{\sigma^2 + \sigma_c^2}$
Complexity: $\mathcal{O}(mn)$

- **Sieve-code-reduce** is an improvement of *Code-reduce* that incorporates lattice sieving to control the progressively increasing coding noise [21]. Since this reduction technique operates similarly to *Code-reduce*, with sieving introduced solely to manage the noise in reduced positions and keep it approximately constant, we will not discuss it separately in the following sections.
- **LMS-reduce** employs lazy modulus switching [3] to reduce an LWE instance modulo q to a scaled instance modulo p, where $p \leq q$. This method accumulates both original noise and rounding noise for LWE instances. Since this technique is primarily effective for small secrets in $\{0, 1\}^n$ or $\{-1, 0, 1\}^n$, and our focus is on uniformly distributed secrets $\mathbf{s} \in \mathbb{Z}_q^n$, we will not consider it further. Improvements to the lazy modulus switching technique are discussed in [10, 27].
- **Guess-reduce** allows for the random guessing of b positions of the secret vector, typically used between the reduction and solving phases to balance their time complexities. By iterating over all possible values with absolute values less than d, $(2d+1)^b$ candidates are generated. This technique does not change the number of samples or the noise level, but impacts the overall time complexity of the BKW algorithm, as it must account for the probability of correctly guessing these secret values.

Guess-reduce: $n' = n - b;\ m' = m;\ \sigma' = \sigma;\ \sigma'_s = \sigma_s$
Complexity: $\mathcal{O}(m \cdot b \cdot (2d+1)^b)$

- **Discard-reduce** reduces b randomly selected dimensions of the sample matrix $\mathbf{A}$ by discarding samples with non-zero values in these dimensions. This technique was originally employed to solve LPN problems, and we extend it to our BKW algorithm for solving LWE problems, providing an analysis of it in the LWE context. The probability of a value being zero in a given dimension is $1/q$, leading to an expected $\frac{m}{q^b}$ samples with all zeros in these b dimensions. Samples are created on-the-fly, and once a non-zero value is detected, the sample can be immediately discarded. This reduction technique does not introduce additional noise.

Discard-reduce: $n' = n - b;\ m' = \frac{m}{q^b};\ \sigma' = \sigma;\ \sigma'_s = \sigma_s$
Complexity: $\mathcal{O}(m(1 + \frac{1}{q} + \cdots \frac{1}{q^{b-1}}))$

We have introduced all existing reduction techniques for the LWE-solving BKW algorithm. Generally, the reduction phase aims to minimize noise while preserving the number of samples. We will explore how to efficiently integrate these techniques within the BKW algorithm in the following sections.

3.2 Solving Techniques

Assume that there are M new LWE samples with a reduced dimension of n' and a noise standard deviation of σ_f. We will now introduce several solving techniques, optimize their analysis, and address inaccuracies in previous literature.

- **Optimal solver** relies on exhaustive search [2]. As discussed in Sect. 2.3, to distinguish the distribution of $\tilde{e}_i$ for a given candidate $\tilde{\mathbf{s}}$, it calculates the log-likelihood ratio using the Neyman-Pearson lemma [34]. Due to its high time complexity $\mathcal{O}(M \cdot q^{n'})$, this technique is rarely used in modern BKW algorithms, and thus we will not provide a detailed explanation here.
- **FFT solver** is an efficient solving technique based on the Fast Fourier Transform (FFT) [16]. Recall Equations (5) and (6), consider the function $f(\mathbf{x}) = \sum_{i=1}^{M} \mathbf{1}_{\mathbf{a}_i = \mathbf{x}} \theta_q^{z_i}$, where $\theta_q := e^{\frac{2\pi i}{q}}$ and $\mathbf{1}_{\mathbf{a}_i=\mathbf{x}}$ is an indicator function that equals 1 if $\mathbf{a}_i = \mathbf{x}$ and 0 otherwise. The key idea is to compute the FFT of $f(\mathbf{x})$:

$$\hat{f}(\boldsymbol{\alpha}) = \sum_{\mathbf{x} \in \mathbb{Z}_q^{n'}} f(\mathbf{x}) \theta_q^{-\langle \mathbf{x}, \boldsymbol{\alpha} \rangle} = \sum_{i=1}^{M} \theta_q^{-(\langle \mathbf{a}_i, \boldsymbol{\alpha} \rangle - z_i)}. \tag{8}$$

If $\boldsymbol{\alpha}$ is the correct candidate $\bar{\mathbf{s}}$, the real part of this function will be relatively large; otherwise, it approaches zero. Thus, with enough samples M, the secret entries can be recovered by maximizing $\mathrm{Re}(\hat{f}(\boldsymbol{\alpha}))$.
The time complexity is $\mathcal{O}(M + n' \cdot q^{n'} \cdot \log(q))$. The sample size required for successful secret recovery is given by $M = 4 \cdot \log(\frac{q^{n'}}{\epsilon}) \cdot (1 - \frac{2\pi^2\sigma^2}{q^2})^{-2^{(n/b)}}$,

where ϵ is the failure probability. We corrected Duc's [16] original estimation $M = 8 \cdot n' \cdot \log(\frac{q}{\epsilon}) \cdot (1 - \frac{2\pi^2\sigma^2}{q^2})^{-2^{(n/b)}}$, fixing inaccuracies in the log term and optimizing the constant factor.

- **Pruned FFT solver** is an improved version of the FFT solver, which limits the number of hypotheses. This technique guesses potential candidates for $\bar{\mathbf{s}}$ with absolute values less than d and identifies the correct candidate similarly to the FFT solver. Due to its efficiency, this technique is utilized in the solving phase of our BKW algorithm.

Pruned FFT solver: $M = 4 \cdot \log(\frac{(2d+1)^{n'}}{\epsilon}) \cdot (1 - \frac{2\pi^2\sigma^2}{q^2})^{-2^{(n/b)}}$
Complexity: $\mathcal{O}(M + n' \cdot q^{n'} \cdot \log(2d+1))$

After presenting the reduction and solving techniques within a unified framework, the BKW algorithm proceeds as follows. Given m initial LWE samples of dimension n, several reduction techniques are applied to generate M new samples with a reduced dimension n'. A solving technique is then employed to recover the n'-dimensional secret with a failure probability of ϵ. This process is iterated with updated samples until the entire secret is recovered.

The time complexity of every reduction technique is determined solely by m, n, and the number of steps, independent of final noise. The sample size needed to recover the secret depends on the final noise. This paper focuses on recovering only a part of the secret, as in prior works [2,3,8,23]. This is because recovering the first part of the secret dominates the overall complexity, while the rest is easier to solve.

4 Optimized BKW Algorithm for LWE

This section presents the graph construction for our optimized BKW algorithm to solve the LWE problem. The graph $G = (V, E)$ is built according to the transition rules of a finite automaton, extended from Bogos's work on LPN [9] to the context of the LWE problem. In this framework, various reduction techniques are represented as edges, while the states of the LWE samples correspond to the vertices. Thus, finding the optimal BKW algorithm is equivalent to identifying the optimal path in this graph. We automate the search for the optimal reduction path in order to determine the most effective combination of techniques, thereby minimizing the time complexity for different LWE instances.

The vertex set $V = \{1, \ldots, n\} \times S$ represents the states of LWE samples, including their dimensions and $\log(\cdot)$ of the number of samples. For instance, the vertex $(n, \log_2 m)$ denotes an LWE instance with dimension n and m samples. The set S can be $\mathbb{R}$, $\mathbb{N}$, or $\{0, \ldots, \eta\}$, where η denotes an upper bound.

The edge set E comprises the reduction techniques applicable between vertices. Each technique is represented as an edge from an $(n, \log_2 m)$ instance to a new instance $(n', \log_2 m')$, where $n' \leq n$ indicates a reduction in dimension. The number of samples may either increase or decrease, along with changes in the noise standard deviation. An edge from $(n, \log_2 m)$ to $(n', \log_2 m')$ specifies the applied reduction technique and the corresponding parameters, such as *LF1-reduce*(b).

In this representation, the reduction phase of the BKW algorithm is viewed as a path in graph G, where each edge corresponds to a reduction technique that progressively reduces the dimension of the LWE instance. This path ultimately leads to a solver, where the pruned FFT is used to recover some entries of the secret vector.

We define a standard formula to represent the change in the noise standard deviation during the reduction phase of the BKW algorithm for solving LWE. The accumulated noise standard deviation σ' is derived from the initial standard deviation σ, with their relationship given by

$$\sigma'^2 = c_1 \cdot \sigma^2 + c_2, \quad c_1, c_2 \in \mathbb{R}. \tag{9}$$

We define a mapping function $\text{Approx}_S : \mathbb{R} \to S$ that returns the closest element in S to a given $x \in \mathbb{R}$. Specifically, we have:

$$\text{Approx}_S(x) = \begin{cases} \lceil x \rceil_S = \min\{s \in S \mid s \geq x\} \\ \lfloor x \rfloor_S = \max\{s \in S \mid s \leq x\}, \end{cases} \tag{10}$$

where $\lceil x \rceil_S$ represents the smallest element in S that is not less than x, and $\lfloor x \rfloor_S$ represents the largest element in S that is not exceed x.

We select the following reduction techniques as candidate edges for the graph: *LF1-reduce*, *LF2-reduce*, *Transform secret*, *Discard-reduce*, *Guess-reduce*, and *Code-reduce*. After defining the noise formula and mapping function, we formalize the selected reduction techniques as follows:

- *LF1-reduce*(b): $(n, \log_2 m) \to (n - b, \text{Approx}_S(\log_2(m - (q^b - 1)/2)))$, with $c_1 = 2$ and $c_2 = 0$.
- *LF2-reduce*(b): $(n, \log_2 m) \to (n - b, \text{Approx}_S(\log_2(\frac{m^2}{q^b - 1} - \frac{m}{2})))$, with $c_1 = 2$ and $c_2 = 0$.
- *Transform secret*: $(n, \log_2 m) \to (n, \text{Approx}_S(\log_2(m - n)))$, with $c_1 = 1$ and $c_2 = 0$.
- *Discard-reduce*(b): $(n, \log_2 m) \to (n - b, \text{Approx}_S(\log_2(m/q^b))$, with $c_1 = 1$ and $c_2 = 0$.
- *Guess-reduce*(b): $(n, \log_2 m) \to (n - b, \text{Approx}_S(\log_2(m))$, with $c_1 = 1$ and $c_2 = 0$.
- *Code-reduce*(n, n'): $(n, \log_2 m) \to (n', \text{Approx}_S(\log_2(m))$, with $c_1 = 1$ and $c_2 = \sigma_c^2$, where σ_c^2 is the variance of the coding noise introduced by the employed linear code $[n, n']$. Our analysis of coding noise and code length is consistent with the *Code-reduce* approach proposed by Guo et al. [23].

As illustrated in Fig. 4, the construction of the graph $G = (V, E)$ must follow the state transition rules of a finite automaton. In this section, we focus on extending the automaton designed by Bogos [9] for LPN to a version suitable

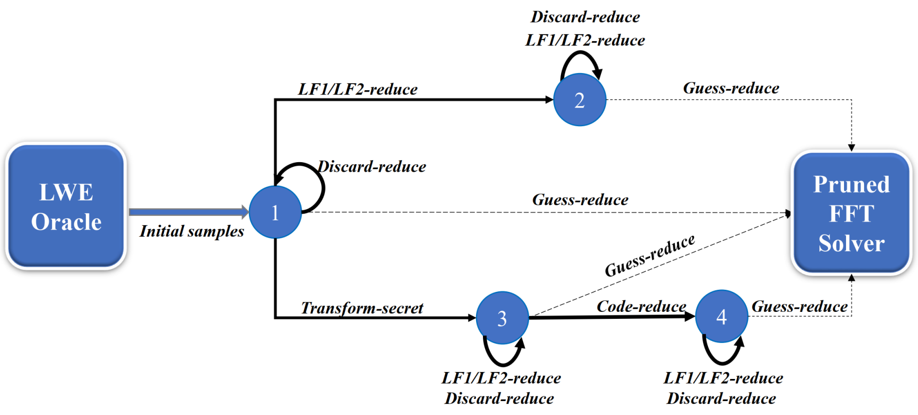

Fig. 4: Finite automaton for our optimized BKW algorithm to solve LWE: a natural extension of Bogos [9] for LPN.

for the LWE problem. In the next section, we will introduce a memory-efficient automaton.

There are several specific constraints on the order of the reduction techniques in the finite automaton. Since the primary purpose of *Transform secret* is to control the standard deviation of the secret, and only *Code-reduce* generates coding noise related to the secret's standard deviation, *Transform secret* must be applied before *Code-reduce*. Additionally, due to the difficulty in analyzing sequences with multiple *Code-reduce* steps, only a single application of *Code-reduce* is permitted. The necessary codes are composed of a cascade of smaller-dimensional codes, as described in coded-BKW [23]. The techniques *LF1-reduce*, *LF2-reduce*, and *Discard-reduce* can be applied multiple times without specific order requirements. *Guess-reduce* is employed to balance the time complexity between the reduction and solving phases and is typically placed at the end of the reduction path, just before the pruned FFT solver.

Based on the finite automaton presented in Fig. 4, we introduce the construction algorithm for the directed graph $G = (V, E)$ in the context of the LWE problem, as described in Algorithm 1. We then provide the definition of the reduction path within this graph.

Definition 2 (Reduction Path). *Consider a graph $G = (V, E)$, where the vertex set is defined as $V = \{(n_0, \log_2 m_0), (n_1, \log_2 m_1), \dots, (n_i, \log_2 m_i)\}$, with $i \in \mathbb{N}$. The edge set $E = \{$LF1/LF2-reduce, Transform secret, Code-reduce, Discard-reduce, Guess-reduce$\}$ represents the available reduction techniques. A sequence*

$$(n_0, \log_2 m_0) \xrightarrow{e_1} (n_1, \log_2 m_1) \xrightarrow{e_2} \dots \xrightarrow{e_i} (n_i, \log_2 m_i) \tag{11}$$

is called a ***reduction path*** *in the BKW algorithm, where each edge $e_i \in E$ corresponds to a specific reduction technique applied at step i.*

Algorithm 1 Construction of the graph for the LWE-solving BKW

```
Require: $n, S, \sigma, \eta, b, q, d$
Ensure: Graph $G = (V, E)$ includes all reduction steps whose complexity does not
    exceed $2^{\eta}$. The vertex set $V = \{1, \ldots, n\} \times S \times state$, where $state = \{1, 2, 3, 4\}$
    corresponds to the state points in the automaton. The edge set $E$ includes all edges
    $((i, \eta_1, st), (j, \eta_2, st'))$ labeled by $(c_1, c_2, r)$, where a $st \xrightarrow{r} st'$ transition exists in the
    automaton and for
 1: $r =$ LF1-reduce:
 2: for all $(i, \eta_1)$ with $rop \leq \eta$ do
 3:     set the edge with $(j, \eta_2) = (i - b, \mathrm{Approx}_S(\log_2(2^{\eta_1} - (q^b - 1)/2)))$
 4:     $c_1 = 2, c_2 = 0, rop = \log_2 i + \eta_1$
 5: end for
 6: $r =$ LF2-reduce:
 7: for all $(i, \eta_1)$ with $rop \leq \eta$ do
 8:     set the edge with $(j, \eta_2) = (i - b, \mathrm{Approx}_S(\log_2(2^{2\eta_1}/(q^b - 1) - 2^{\eta_1}/2)))$
 9:     $c_1 = 2, c_2 = 0, rop = \log_2 i + \max(\eta_1, \eta_2)$
10: end for
11: $r =$ Discard-reduce:
12: for all $(i, \eta_1)$ with $rop \leq \eta$ do
13:     set the edge with $(j, \eta_2) = (i - b, \mathrm{Approx}_S(\eta_1 - \log_2(q^b))))$
14:     $c_1 = 1, c_2 = 0, rop = \log_2 i + \eta_1 + \log_2((q - 1/q^b)/(q - 1))$
15: end for
16: $r =$ Transform-secret:
17: for all $\eta_1 : 1$ with $rop \leq \eta$ do
18:     set the edge with $i = n$, $(j, \eta_2) = (i, \mathrm{Approx}_S(\log_2(2^{\eta_1} - i)))$
19:     $c_1 = 1, c_2 = 0, rop = \log_2(i^2 + ((2^{\eta_1} - i) \cdot i^2)/(\log_2 i - \log_2 \log_2 i))$
20: end for
21: $r =$ Guess-reduce:
22: for all $(i, \eta_1)$ with $rop \leq \eta$ do
23:     set the edge with $(j, \eta_2) = (i - b, \mathrm{Approx}_S(\eta_1))$
24:     $c_1 = 1, c_2 = 0, rop = \eta_1 + \log_2 b + b \cdot \log_2(2d + 1)$
25: end for
26: $r =$ Code-reduce:
27: for all $(i, \eta_1, j)$ with $j < i$ and $rop \leq \eta$ do
28:     set the edge with $\eta_2 = \eta_1, c_1 = 1, c_2 = \sigma_c^2, rop = \log_2 i + \eta_1$, where $\sigma_c$ is the
        coding noise standard deviation from code $[i, j]$
29: end for
```

A reduction path is considered **valid** if it follows the order of reduction techniques illustrated in the automaton and terminates at the solver. If a valid reduction path ends at the state $(n_t, \log_2 m_t)$, which is accepted by the pruned FFT solver, then the remaining n_t positions of the secret will be recovered with high probability. We then define the valid reduction path formally.

Definition 3 (Valid Reduction Path). *Consider a reduction path in the graph $G = (V, E)$ constructed by Algorithm 1:*

$$(n_0, \log_2 m_0) \xrightarrow{e_1} (n_1, \log_2 m_1) \xrightarrow{e_2} \ldots \xrightarrow{e_i} (n_i, \log_2 m_i), \tag{12}$$

where each edge $e_j = (c_{1,j}, c_{2,j}, r)$, with r representing the applied reduction technique, and c_1, c_2 are coefficients that affect the noise standard deviation.

Let σ_j denote the noise standard deviation at the vertex $(n_j, \log_2 m_j)$, defined recursively by $\sigma_0 = \sigma$, and

$$\sigma_j^2 = c_{1,j}\sigma_{j-1}^2 + c_{2,j}, \quad \textit{for } j = 1, \ldots, i. \tag{13}$$

The path is considered an ***ϵ-valid reduction path*** *if m_i meets the sample requirements for successful secret recovery by the pruned FFT solver, where ϵ is the failure probability.*

Using the above algorithm and definitions, we can construct the graph for our optimized BKW algorithm, based on the automaton tailored for the LWE problem. This approach allows us to formalize the reduction phase of the BKW algorithm as a search for the optimal valid reduction path to solve various LWE instances. Our goal is to identify the most efficient valid combination of reduction techniques (i.e., the reduction path). The detailed search algorithm is presented in Sect. 5.3.

5 Memory-Efficient BKW Algorithm

The primary limitation of the BKW algorithm for solving LWE (or LPN) is its substantial memory requirement, making it impractical even for medium-sized parameters. We found that the natural extension of Bogos's work [9] on LPN to LWE does not alleviate the significant memory consumption of the BKW algorithm. For example, as illustrated in Fig. 4, the *Discard-reduce* technique can be repeatedly applied at each state, which is likely to result in significant memory overhead. Moreover, all reduction techniques in Fig. 4 rely on traditional pairwise reduction (i.e., 2-sum), without considering modern memory-saving methods such as multiple collision finding, i.e., c-sum reduction techniques for $c \geq 2$.

This section presents a memory-efficient modification of the natural extension of the BKW algorithm introduced in Sect. 4. Our approach includes two key improvements. First, in Sect. 5.1, we introduce a more efficient reduction technique for low-memory scenarios by combining multiple collision finding with Parallel Collision Search (PCS). Additionally, we propose a memory-efficient automaton in Sect. 5.2 by pruning high-memory paths, integrating memory-saving reduction techniques, and rearranging the order of reduction methods. Finally, we present algorithms for searching the optimal reduction path in our optimized BKW algorithm to solve various LWE instances in Sect. 5.3.

5.1 Reduction Techniques for Low-Memory Regimes

In this section, we introduce several efficient reduction techniques tailored for LWE-solving BKW algorithm, based on multiple collision finding, i.e., c-sum for $c \geq 2$. Inspired by Delaplace's work [12], we propose a novel reduction technique, c-sum-PCS-reduce, which integrates Parallel Collision Search (PCS) to improve efficiency in low-memory scenarios. This technique enriches the existing c-sum techniques extended from LPN to LWE. Finally, we compare the performance of these c-sum techniques in terms of time-memory trade-offs within the context of the LWE problem.

Two c-sum techniques used in LPN-solving BKW algorithms can be directly extended to LWE-solving BKW algorithms [17]. The original BKW algorithm [7] solves the LPN problem with time and memory complexities of $2^{\frac{n}{\log n}(1+o(1))}$ using 2-sum. Esser et al. [17] proposed the c-sum technique, termed c-sum-naive-reduce, to balance the time and memory complexities. This technique identifies c-tuples of vectors that sum to zero at selected b positions. By leveraging the exponential growth of size-c subsets, it significantly reduces the number of initial samples needed, thereby lowering memory usage. They further improved the time-memory trade-off for the LPN-solving BKW algorithm by combining c-sum with a dissection technique, which asymmetrically decomposes the problem and solves it recursively, referred to as c-sum-dissect-reduce.

We now present the c-sum-PCS-reduce technique for the LWE-solving BKW algorithm, designed to improve efficiency in low-memory settings. This technique combines the c-sum technique with PCS [39, 40], which has been widely studied for enhancing cryptanalysis algorithms [14, 19, 35]. The main idea involves splitting the c-sum problem into two symmetric $\frac{c}{2}$-sum problems and using PCS to detect sample collisions.

First, we formally define the c-sum problem over the finite field $\mathbb{F}_q$, where q is an arbitrary prime modulus. Since $\mathbb{Z}_q$ and $\mathbb{F}_q$ are equivalent for prime q, we use $\mathbb{F}_q$ in our definitions for the LWE problem to align with the LPN problem.

Definition 4 (c-Sum Problem (c-SP)). *Consider positive integers b, N and $c \geq 2$. Define a list $L := \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, where each element $\mathbf{x}_i$ is uniformly drawn from $\mathbb{F}_q^b$. A single-solution of the c-SP$_b$ problem is a subset $\mathcal{L} \in \binom{[N]}{c}$ of size c where*

$$\sum_{j \in \mathcal{L}} \mathbf{x}_j = 0^b. \tag{14}$$

Given a list L, c-sum problem c-SP$_b$ involves determining at least N unique single-solutions. Thus, the goal is to identify N distinct c-sized subsets of b-dimensional vectors in L, with their sum equal to the zero vector.

We extract the c-sum problem from the reduction phase, since solving this problem dominates the overall complexity of the c-sum-BKW algorithm for LWE, as shown in the following theorem. Furthermore, we adopt the independence heuristic proposed by Esser et al. [17], treating the c-sum problem as independent in our analysis.

Theorem 1 (Complexities of c-sum-BKW [17]). *Let $b, c, N \in \mathbb{N}$. Under the independence heuristic, an algorithm solving the c-SP$_b$ problem for an input list of size N with expected time T and memory M can also solve the n-dimensional LWE problem with high probability. This requires time $T^{1+o(1)}$, memory $M^{1+o(1)}$, and $N^{1+o(1)}$ samples, provided $N \geq q^{\frac{b+c\log_q c+1}{c-1}}$.*

To satisfy the lower bound in Theorem 1, we set the list size $N = |L| = c \cdot q^{\frac{2b}{c-1}} = q^{\frac{2b+c\log_q c-\log_q c}{c-1}} > q^{\frac{b+c\log_q c+1}{c-1}}$. We assume c is an even positive integer for simplicity. We divide L into c equal-sized sublists, each containing $|L_i| = q^{\frac{2b}{c-1}}$ elements, where $i = 1, \ldots, c$. Each sublist L_i is viewed as a set over $\mathbb{F}_q^{\frac{2b}{c-1}}$. we then randomly select one element from each sublist, denoting the k^{th} element of sublist L_i as $L_i[k]$.

To facilitate the application of PCS, we define two mapping functions. For $j = 0, 1$, we set

$$\begin{aligned} f_j : (\mathbb{F}_q^{\frac{2b}{c-1}})^{\frac{c}{2}} &\to \mathbb{F}_q^b, \\ (l_1, \ldots, l_{\frac{c}{2}}) &\mapsto L_{j\frac{c}{2}+1}[l_1] + L_{j\frac{c}{2}+2}[l_2] + \cdots + L_{\frac{c}{2}(1+j)}[l_{\frac{c}{2}}]. \end{aligned} \tag{15}$$

The mapping function f_0 maps $\frac{c}{2}$ indices to a $\frac{c}{2}$-sum problem defined on the first half of all sublists, while f_1 does the same for the second half. Consequently, if a collision occurs between f_0 and f_1, i.e., $f_0(l_1, \ldots, l_{\frac{c}{2}}) = f_1(l_{\frac{c}{2}+1}, \ldots, l_c)$, we obtain a c-sum problem that satisfies

$$L_1[l_1] + \cdots + L_c[l_c] = 0^b. \tag{16}$$

This represents a single-solution to the c-sum problem. To fully resolve this problem, we need to identify $N = c \cdot q^{\frac{2b}{c-1}}$ distinct collisions between f_0 and f_1 using the procedure PCS(f_0, f_1, N). It is commonly assumed that the collisions returned by PCS are uniformly random, as discussed in [12, 13].

For example, we consider $c = 4$ and $c = 8$ and demonstrate a single reduction iteration that reduces b dimensions, as shown in Fig. 5.

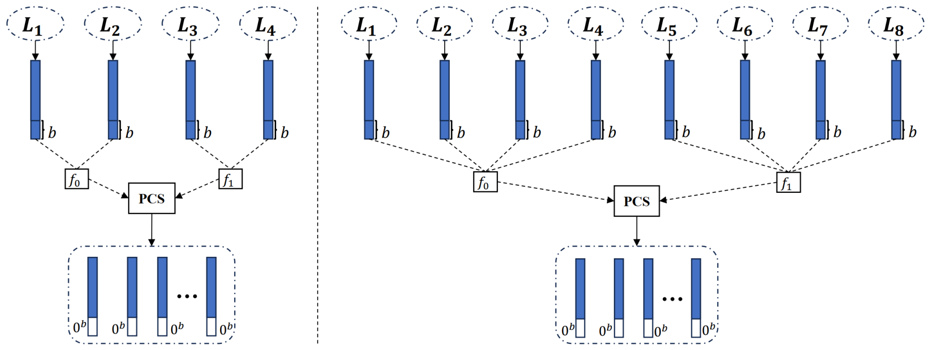

Fig. 5: One iteration of the c-sum-PCS-reduce for $c = 4$ and $c = 8$.

Theorem 2 (Complexities of c-sum-PCS-reduce). *Given an even positive integer c, the size of a c-SP$_b$ instance L is $|L| = N := c \cdot q^{\frac{2b}{c-1}}$. The c-sum-PCS-reduce, under independence heuristic, solves c-SP$_b$ with an expected time complexity of $T = \tilde{O}(q^{(\frac{1}{2}+\frac{1}{c-1})b})$ and memory $M = \tilde{O}(q^{\frac{b}{c-1}})$.*

Proof. The complexity of the c-sum-PCS-reduce primarily depends on the complexity of the PCS procedure. We first analyze the complexity of PCS over the q-ary field. Consider two random functions $f_0, f_1 : \mathbb{F}_q^r \to \mathbb{F}_q^r$, where $r, q \in \mathbb{N}$. f_0 and f_1 are treated as independent random functions under the independence heuristic. The procedure $\mathrm{PCS}(f_0, f_1, N)$ is expected to find N distinct collisions between f_0 and f_1 with a time complexity of $T = \tilde{\mathcal{O}}(q^{\frac{r+\log_q N}{2}})$ and a memory complexity of $M = \tilde{\mathcal{O}}(N)$.

During each reduction iteration, we only consider b positions and aim to find $N = c \cdot q^{\frac{2b}{c-1}}$ collisions between f_0 and f_1. Consequently, the memory required for the c-sum-PCS-reduce is $M = \tilde{O}(N) = \tilde{O}(q^{\frac{2b}{c-1}})$. The expected time complexity can be formulated as:

$$T = \tilde{O}(q^{\frac{b+\log_q N}{2}}) = \tilde{O}(q^{\frac{b}{2}} \cdot q^{\frac{1}{2}\cdot\left(\frac{2b+c\log_q c - \log_q c}{c-1}\right)}) = \tilde{O}(q^{\frac{b}{2}} \cdot q^{\frac{b}{c-1}}) = \tilde{O}(q^{\left(\frac{1}{2}+\frac{1}{c-1}\right)b}).$$

□

The BKW algorithm corresponding to the c-sum-PCS-reduce is called the c-sum-PCS-BKW algorithm. To analyze the complexity of this algorithm, it is essential to select an appropriate value for the block size b.

Define the modulus of the n-dimensional LWE problem as $q = \mathcal{O}(n^{c_q})$ and the noise standard deviation as $\sigma = \mathcal{O}(n^{c_s})$, where $c_q > c_s > 0$ are constants. As discussed in [26], the original BKW algorithm uses a block size b with a blocks where $ab = n$, and selects $a = (1 + 2c_q - 2c_s) \cdot \log n$. The time complexity and memory complexity of the original BKW algorithm are as follows:

$$q^{b(1+\epsilon)} = 2^{\frac{1}{2}\cdot\frac{c_q}{c_q - c_s + \frac{1}{2}}\cdot n(1+\epsilon)}, \text{ where } \epsilon > 0. \tag{17}$$

Since the c-sum-PCS-BKW algorithm combines c samples in each reduction to generate new ones, we adapt the approach from [18] by modifying a with a factor of $\log c$. Specifically, the adjusted value is defined as:

$$a = (1 + 2c_q - 2c_s) \cdot \frac{\log n}{\log c}. \tag{18}$$

Consequently, the block size b becomes:

$$b = \frac{n}{a} = \frac{n \cdot \log c}{(1 + 2c_q - 2c_s) \cdot \log n}. \tag{19}$$

By combining Theorems 1 and 2 with the value of b, we derive the following complexity theorem for the c-sum-PCS-BKW algorithm.

Theorem 3 (Complexities of c-sum-PCS-BKW). *Given an even positive integer c, and $\epsilon > 0$. Under the independence heuristic, the LWE problem with parameters $(n, q = \mathcal{O}(n^{c_q}), \sigma = \mathcal{O}(n^{c_s}))$ can be solved with high probability by the c-sum-PCS-BKW algorithm with time*

$$T = 2^{\left(\frac{1}{2}+\frac{1}{c-1}\right)\cdot n\cdot\log c\cdot\frac{1}{2}\cdot\frac{c_q}{c_q-c_s+\frac{1}{2}}\cdot(1+\epsilon)} \textit{ and memory } M = 2^{\frac{2}{c-1}\cdot n\cdot\log c\cdot\frac{1}{2}\cdot\frac{c_q}{c_q-c_s+\frac{1}{2}}\cdot(1+\epsilon)}.$$

Proof. By substituting the value of b from Equation (19) into the complexity expression of Theorem 2 for the c-sum-PCS-reduce, we derive the complexity formula for the c-sum-PCS-BKW algorithm. □

Comparison of Different c-sum Techniques We compare our proposed c-sum-PCS-reduce with other existing c-sum techniques, including c-sum-naive-reduce, c-sum-dissect-reduce, and c-sum-quantum-reduce, in the context of solving the LWE problem. To facilitate this comparison, we first present the time and memory complexities of the corresponding c-sum-BKW in Table 1, followed by a comparison of their time-memory trade-offs in Table 2.

We express the complexities of the different c-sum-BKW algorithms in a unified format as follows:

$$T = 2^{x(1+\epsilon)}, \quad M = 2^{y(1+\epsilon)}, \tag{20}$$

where

$$x = t \cdot n \cdot \frac{1}{2} \cdot \frac{c_q}{c_q - c_s + \frac{1}{2}}, \quad y = m \cdot n \cdot \frac{1}{2} \cdot \frac{c_q}{c_q - c_s + \frac{1}{2}}. \tag{21}$$

The values of t and m are presented in Table 1.

Table 1: Partial complexity exponents t and m for various c-sum-BKW algorithms solving the LWE problem.

	Algorithm	t	m	for
classic	Original BKW [2]	1	1	
	c-sum-naive-BKW [18]	$\log c$	$\frac{\log c}{c-1}$	$c \geq 2$
	c-sum-dissect-BKW [18]	$(1 - \frac{i}{c-1}) \cdot \log c$	$\frac{\log c}{c-1}$	$c \in magic$
	c-sum-PCS-BKW	$(\frac{1}{2} + \frac{1}{c-1}) \cdot \log c$	$\frac{2 \cdot \log c}{c-1}$	$c \geq 2$
quantum	c-sum-quantum-BKW [18]	$\frac{1}{2} \cdot \frac{c}{c-1} \cdot \log c$	$\frac{\log c}{c-1}$	$c \geq 2$

* *magic* represents a sequence that satisfies $c_i = c_{i-1} + i + 1$, with $c_{-1} = 1, i \in \mathbb{N} \cup \{0\}$.

We derive the time-memory trade-offs for various c-sum-BKW algorithms based on the results in Table 1. These trade-offs are summarized in Table 2 using the most commonly used time-memory trade-off notation. To further illustrate these comparisons, we plot the corresponding time-memory trade-off curves for each algorithm in Fig. 1, based on the formulas in Table 2.

As shown in Table 2, the c-sum-BKW algorithms that incorporate various c-sum reduction techniques offer significant advantages in time-memory trade-offs compared to the original BKW algorithm, with the best trade-off observed in the c-sum-quantum-BKW algorithm. In comparison, our c-sum-PCS-BKW

Table 2: Time-memory trade-off comparisons of various c-sum-BKW algorithms for solving LWE: our c-sum-PCS-BKW vs. other c-sum-BKW algorithms.

	Trade-offs	$2^{n\cdot\log c\cdot\frac{1}{2}\cdot\frac{c_q}{c_q-c_s+\frac{1}{2}}\cdot(1+\epsilon)} =$	for
classic	Original BKW [2]	$T^{\frac{1}{2}}\cdot M^{\frac{1}{2}}$	
	c-sum-naive-BKW [18]	$T^{\frac{1}{2}}\cdot M^{\frac{c-1}{2}}$	$c \geq 2$
	c-sum-dissect-BKW [18]	$T\cdot M^{i}$	$c \in \mathit{magic}$
	c-sum-PCS-BKW	$T\cdot M^{\frac{c-3}{4}}$	$c \geq 2$
quantum	c-sum-quantum-BKW [18]	$T\cdot M^{\frac{c-2}{2}}$	$c \geq 2$

* *magic* represents a sequence that satisfies $c_i = c_{i-1}+i+1$, with $c_{-1} = 1, i \in \mathbb{N}\cup\{0\}$.

algorithm requires approximately the square of the memory used by the c-sum-quantum-BKW algorithm for the same time complexity. The comparison between our c-sum-PCS-BKW and the c-sum-dissect-BKW is less apparent in Table 2, but the trade-off curves in Fig. 1 provide a clearer comparison.

Fig. 1 highlights the time-memory trade-offs of our c-sum-PCS-BKW algorithm compared to other c-sum-BKW algorithms. When memory is constrained to $2^{0.3n}$, our algorithm shows lower memory consumption than the classical c-sum-dissect-BKW algorithm for the same time complexity. Additionally, for the same memory complexity, c-sum-PCS-BKW achieves lower time complexity. Our algorithm offers substantial advantages in both time and memory complexity when memory is limited to $2^{0.63n}$, compared to the classical c-sum-naive-BKW algorithm.

5.2 Improved Memory-Optimized Finite Automaton

This section presents improvements to the finite automaton designed in Sect. 4 for our optimized LWE-solving BKW algorithm, resulting in a memory-optimized finite automaton. We proved the optimal order of reduction techniques in the LWE scenario, inspired by Wiggers [41] but extended to a more general framework suitable for larger moduli and noise levels.

First, we introduce three classic c-sum reduction techniques—c-sum-PCS-reduce, c-sum-dissect-reduce, and c-sum-naive-reduce—into the original finite automaton described in Fig. 4. The reduction techniques in the original automaton rely solely on traditional pairwise reduction (i.e., 2-sum) and do not incorporate modern c-sum techniques. We have provided a detailed explanation of the advantages of c-sum reduction techniques in low-memory regimes in Sect. 5.1. We then present the improved automaton with the newly added reduction techniques in Fig. 6. To simplify the representation of the automaton, we use *Reduce-techniques* to denote the available reduction methods, including various c-sum techniques, *LF1-reduce*, and *LF2-reduce*.

Second, we prune the potential high-memory paths in the original automaton shown in Fig. 4. We observe that, under memory constraints, the original

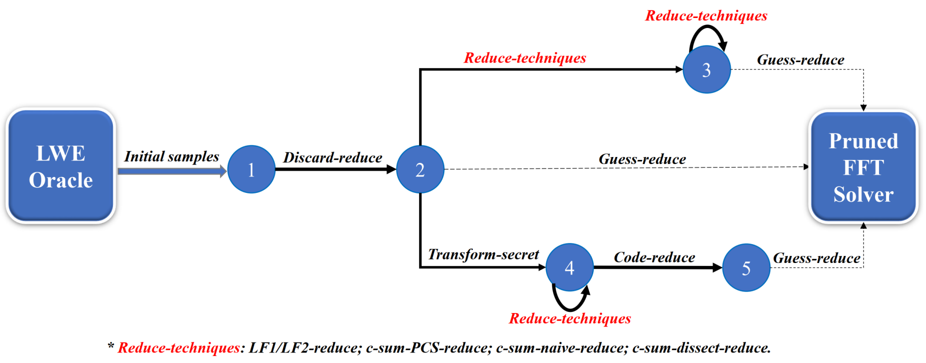

Fig. 6: Improved memory-optimized automaton for our BKW algorithm to solve the LWE problem.

automaton becomes significantly simplified because of many impossible paths. For example, in the original automaton, the *Discard-reduce* technique applies after each reduction technique and is repeatedly used at each state, which may lead to high memory consumption. However, we present the following Proposition 1 to demonstrate that, in memory-constrained scenarios, no technique from *Reduce-techniques* $\rightarrow$ *Discard-reduce* can occur in the optimal reduction path of our optimized BKW algorithm to solve the LWE problem.

Proposition 1. *No technique from Reduce-techniques $\rightarrow$ Discard-reduce can occur in the optimal reduction path for solving the LWE problem under memory constraints.*

Proof. We take *LF1-reduce* as an example to prove that the order *LF1-reduce* $\rightarrow$ *Discard-reduce* cannot occur under memory constraints, and similar proofs apply to other techniques within the *Reduce-techniques*. Our proof is inspired by Wiggers et al. [41].

Assume that we need to reduce the LWE problem with dimension n. Utilizing the order *LF1-reduce* $\rightarrow$ *Discard-reduce*, we first reduce it to LWE with dimension $n-b$ via *LF1-reduce*, and then to LWE with dimension n' using *Discard-reduce*, where $b \in [0, n-n']$. This sequence takes time $T \approx (2n-b) \cdot (m - \frac{q^b-1}{2})$ and memory $M \approx \max\{mn, (m - \frac{q^b-1}{2})(n-b)\}$. Both functions $T(b)$ and $M(b)$ are strictly decreasing in b, achieving their minimum at $b = n - n'$. Notably, the number of remaining samples is independent of b, meaning that b does not impact subsequent reductions. In summary, in the optimal path, any order *LF1-reduce* $\rightarrow$ *Discard-reduce* simplifies to *LF1-reduce*. □

We have also considered the orders *Code-reduce* $\rightarrow$ *Discard-reduce* and *Code-reduce* $\rightarrow$ any technique within *Reduce-techniques*. No further reduction technique after *Code-reduce* is allowed due to the complex relationship between time complexity and the coding noise introduced by the code. Consequently, we prune the reusable *Discard-reduce* technique from states 1, 2, 3, and 4 in the original automaton shown in Fig. 4, as well as the reusable *LF1/LF2-reduce* techniques from state 4.

Finally, we modified the original automaton by adjusting the order of *Discard-reduce*, placing it as the first step. We support on-the-fly sample generation, and once a non-zero value is detected, the sample can be immediately discarded. Only samples that meet the *Discard-reduce* criteria can remain in memory. This fits well in a memory-constrained scenario, such as in the LPN settings [41].

We present the improved memory-optimized finite automaton in Fig. 6, which allows us to reconstruct graphs for our optimized BKW algorithm to solve LWE, using a method similar to that in Sect. 4.

5.3 Finding the Optimal Reduction Path

Building on Bogos's work [9], we propose two algorithms tailored for the LWE setting to identify optimal valid reduction paths for the BKW algorithm to solve various LWE instances.

The time complexity of BKW, following the reduction path $(e_1, \ldots, e_i)$, is determined by the total complexity of each reduction step and the pruned FFT solver. We approximate BKW's complexity by taking the maximum of these values, defining it as the **max-complexity** of the path.

Note that if the set $S = \mathbb{R}$, i.e., Approx_S is the identity function, such a reduction path is called an *exact path*; otherwise, it is called a *rounded path*. Since exact paths are computationally expensive in practice, and rounded paths offer a good balance between performance and computational cost, we aim to search for an optimal *rounded path* that minimizes *max-complexity* within a required precision.

We first introduce Algorithm 2, which is designed to find the optimal valid reduction path for the optimized BKW algorithm to solve LWE, ensuring that its max-complexity is bounded by 2^η.

As illustrated in Algorithm 2, this algorithm inputs the parameters of the LWE problem, the failure probability ϵ associated with recovering the secret vector, and an upper limit η on the logarithmic time complexity of each reduction step. It begins by constructing a directed graph G following the framework outlined in Algorithm 1, which includes all feasible reduction steps with a time complexity below 2^η. It is important to note that we do not directly call Algorithm 1 to store all edges; instead, we generate the edges based on the rules specified in the finite automaton in Fig. 6.

For each vertex, Algorithm 2 iteratively defines $\texttt{Edge}^{st}$ and Σ^{st}, which denote the optimal reduction step to reach that vertex and the associated noise variance, respectively. We aim to identify the reduction path that minimizes noise variance. Ultimately, we arrive at a state that can be processed by the pruned FFT solver, ensuring that the complexity of the solving phase remains within 2^η. By retracing our steps through this process, we construct the optimal valid rounded reduction path, with the logarithmic max-complexity constrained by η.

Algorithm 2 Search for an optimal valid rounded reduction path with max-complexity bounded to η

```
Require: $n, \sigma, S, \eta, q, b, \epsilon$
Ensure: An optimal valid rounded path with max-complexity bounded to $\eta$
 1: Construct graph $G$ using Algorithm 1 with parameters $n, \sigma, S, \eta, q, b, d$
 2: for all $\eta_1 \in S$ do
 3:     $\Sigma^0_{n,\eta_1} \leftarrow \sigma^2$, $\texttt{Edge}^0_{n,\eta_1} \leftarrow \perp$, $\Sigma^{st}_{n,\eta_1} \leftarrow -\infty$, $\texttt{Edge}^{st}_{n,\eta_1} \leftarrow \perp$    ▷ $\Sigma^{st}_{n,\eta_1}$ stores the noise standard deviation for a vertex $(n, \eta_1, st)$ in a path, and $\texttt{Edge}^{st}_{n,\eta_1}$ represents the edge reaching this vertex
 4: end for
 5: for $j \leftarrow n$ decreasing to 1 do
 6:     for $\eta_2 \in S$ in decreasing order do
 7:         $\Sigma^{st}_{j,\eta_2} \leftarrow 0$, $\texttt{Edge}^{st}_{j,\eta_2} \leftarrow \perp$ for all $st$
 8:         for each $st'$ and each edge $e$ to $(j, \eta_2, st')$ do
 9:             Let $(i, \eta_1, st)$ be $e$'s origin, with $c_1, c_2$ derived from $e$
10:             if $c_1 \Sigma^{st}_{i,\eta_1} + c_2 \leq \Sigma^{st'}_{j,\eta_2}$ then
11:                 $\Sigma^{st'}_{j,\eta_2} \leftarrow c_1 \Sigma^{st}_{i,\eta_1} + c_2$, $\texttt{Edge}^{st'}_{j,\eta_2} \leftarrow e$
12:             end if
13:         end for
14:         if $\eta_2 > 4 \cdot \log(\frac{(2d+1)^j}{\epsilon}) \cdot (1 - \frac{2\pi^2 \Sigma^{st'}_{j,\eta_2}}{q^2})^{-2^{(n/b)}}$ and $j + \log_2 j \leq \eta$ then
15:             Define the path $c$ terminating at $\texttt{Edge}^{st'}_{j,\eta_2}$ and output $(c, \texttt{true})$
16:         end if
17:     end for
18: end for
19: Output $(\perp, \text{false})$
```

While Algorithm 2 is capable of identifying the optimal valid rounded reduction path within a specified logarithmic max-complexity bound η, the challenge of determining the smallest value of η remains unresolved. To tackle this issue, we present a new algorithm, as illustrated in Algorithm 3.

Algorithm 3 efficiently determines the smallest η by employing a divide-and-conquer strategy. By utilizing Algorithm 2 as a subroutine, Algorithm 3 guarantees that Algorithm 2 can locate a valid rounded reduction path with an optimal max-complexity constrained by 2^η. In other words, Algorithm 3 can find a valid rounded reduction path for the LWE-solving BKW algorithm, achieving optimal max-complexity within specified precision.

Specifically, Algorithm 3 utilizes a binary search technique, beginning with an initial value of $\eta = n$ derived from a brute-force result. It then iteratively halves the search space for η, progressively refining the range to quickly determine the smallest valid η. This approach considerably decreases the number of calls to Algorithm 2 compared to a straightforward linear search across the entire η range, thereby enhancing overall search efficiency. Through this method, Algorithm 3 ensures that Algorithm 2 not only identifies a valid rounded path that meets the specified precision but also minimizes its complexity as effectively as possible.

We outline the complexity of the aforementioned algorithms. In Algorithm 1, we construct a graph with $\mathcal{O}(n \cdot |S|)$ vertices, where each vertex connects to $\mathcal{O}(n)$ other vertices, leading to a total graph size of $\mathcal{O}(n^2 \cdot |S|)$. In Algorithm 2, the

Algorithm 3 Algorithm for finding a valid rounded path with optimal max-complexity

```
Require: $n, \sigma, \eta, q, b, \epsilon$, prec $= 0.1$
Ensure: A valid rounded path
1: let found $\leftarrow$ bruteforce, rise $\leftarrow n$, $\eta \leftarrow n$
2: repeat
3:     let rise $\leftarrow$ rise/2
4:     define $S \leftarrow \{0, \text{prec}, 2 \times \text{prec}, \ldots\} \cap [0, \eta - \texttt{rise}]$
5:     $(\text{output'}, \text{success}) \leftarrow \text{Search}(n, \sigma, \epsilon, q, d, b, S, \eta - \texttt{rise})$ with Algorithm 2
6:     if success then
7:         let found $\leftarrow$ output', $\eta \leftarrow \eta - \texttt{rise}$
8:     end if
9: until rise $\leq$ prec
10: output found
```

LWE dimension offers n potential values, while the number of samples can take $|S|$ distinct values. Additionally, for any fixed vertex, each type of edge has $\mathcal{O}(n)$ possible sources. Thus, the complexity of Algorithm 2 is $\mathcal{O}(n^2 \cdot |S|)$, indicating a linear relationship with the size of the graph. The memory complexity of Algorithm 2 is $\mathcal{O}(n \cdot |S|)$, primarily allocated for storing $\Sigma^{n,\eta}$ and $\text{Edge}^{n,\eta}$. Using a binary search approach, Algorithm 3 calls Algorithm 2 approximately $\log \frac{n}{\text{precision}}$ times.

6 Comparison of Estimation Results

6.1 Time Complexity Comparison

This section estimates the time complexity of our optimized BKW algorithm for solving various LWE instances and compares it with the state-of-the-art coded-BKW algorithm [23] from the leading lattice-estimator [4], currently the most effective tool in post-quantum cryptanalysis.

We selected the LWE parameters suggested by Regev [37], Lindner-Peikert [30], and the TU Darmstadt challenge [1], with the comparisons presented in Table 3, respectively. In column 5 of Table 3, the results of the coded-BKW from the lattice-estimator [4] are provided. Columns 6 and 7 show the results of our BKW algorithm, representing the optimal max-complexity ('max-') and the optimal total complexity ('tot-'), respectively. The last column highlights the improvements of our algorithm, calculated as the difference between the total complexity and the values in column 5. The values demonstrate that our optimized BKW algorithm outperforms the previously most effective coded-BKW algorithm [23] for all parameters considered.

Regev [37] defines n as a positive integer, q as the smallest prime greater than n^2, and $\sigma = \frac{q}{\sqrt{n} \cdot \log^2 n \sqrt{2\pi}}$. We select values of n in the range $n = [64, 80, 96, 112, 128, 160, 224, 256, 384, 512]$. Following previous BKW algorithms [2, 3, 16, 23], we set the success probability to 0.99, and assume that operations over $\mathbb{C}$ have the same complexity as those over $\mathbb{Z}_q$. We also assume that the initial samples we provided are sufficient for use.

Table 3: Comparison of estimated time complexity (in $\log_2(\cdot)$) of the BKW algorithm for various LWE instances with parameters from Regev [37], Lindner-Peikert [30], and TU Darmstadt LWE challenges [1].

Parameters	n	q	σ	**Lattice-estimator** [4]	**This paper max-**	**tot-**	**Impro-vements**
Regev [37]	64	4099	5.68	47.5	41.6	**42.2**	**5.3**
	80	6421	7.17	62.8	52.8	**53.4**	**9.4**
	96	9221	8.66	65.3	55.1	**55.5**	**9.8**
	112	12547	10.21	68.8	57.1	**57.5**	**11.3**
	128	16411	11.81	84.5	71.7	**72.0**	**12.5**
	160	25601	15.06	101.5	89.5	**90.2**	**11.3**
	224	50177	21.94	127.9	117.0	**117.0**	**10.9**
	256	65537	25.53	145.1	129.3	**129.7**	**15.4**
	384	147457	40.73	221.4	214.3	**214.3**	**7.1**
	512	262147	57.06	287.6	261.4	**261.4**	**26.2**
Lindner-Peikert [30]	128	2053	2.70	69.7	58.3	**58.5**	**11.2**
	192	4099	8.87	109.6	100.2	**100.5**	**9.1**
	320	4099	8.00	170.3	149.1	**149.4**	**20.9**
	512	4099	2.90	209.2	184.7	**184.8**	**24.4**
LWE Challenge [1]	40	1601	8.01	41.7	32.3	**32.8**	**8.9**
		1601	16.01	41.7	32.2	**32.7**	**9.0**
		1601	24.02	51.7	45.0	**45.0**	**6.7**
	50	2503	12.52	44.5	38.1	**38.9**	**5.6**
		2503	37.55	55.1	46.3	**46.4**	**8.7**
		2503	75.09	65.8	62.5	**62.5**	**3.3**
	70	4903	24.52	60.3	50.9	**51.3**	**9.0**
		4903	49.03	72.1	63.6	**63.6**	**8.5**
		4903	73.55	74.5	65.3	**65.3**	**9.2**
	120	14401	72.01	108.8	97.5	**97.5**	**11.3**
		14401	144.01	124.0	107.0	**107.0**	**17.1**
		14401	216.02	136.8	120.5	**120.5**	**16.3**

We present a detailed analysis of Regev's [37] parameters in Table 3, demonstrating that our optimized BKW algorithm outperforms the state-of-the-art coded-BKW algorithm [4]. Specifically, we achieve an improvement in time complexity ranging from 5.3 to 26.2 bits across all considered parameters. To illustrate how our BKW algorithm finds the optimal reduction path, we provide an example. For an LWE instance with $(n, q, \sigma) = (256, 65537, 25.53)$, the optimal reduction path searched by our algorithm is:

$$(256, 228.13) \xrightarrow{\textcircled{1}} (249, 116.13) \xrightarrow{\textcircled{2}} (249, 116.13) \xrightarrow{\textcircled{3}} (109, 114.91) \xrightarrow{\textcircled{4}} (8, 112.59) \xrightarrow{\textcircled{5}} (7, 112.59) \rightarrow \text{Solver}.$$

Here, $\textcircled{1}$ represents *Discard-reduce*(7), $\textcircled{2}$ represents *Transform-secret*, $\textcircled{3}$ involves 20 iterations of *LF1-reduce*(7), $\textcircled{4}$ represents *Code-reduce*(101), and $\textcircled{5}$ represents *Guess-reduce*(1). The *Code-reduce* uses a $[101, 84]$ concatenated code, composed of 12 random linear lattice codes: one $[11, 7]$, two $[10, 7]$, two $[9, 7]$, three $[8, 7]$, and four $[7, 7]$. After the reduction, the new LWE instance with dimension 7 is passed to the solver to recover the secret subvector. Ultimately, our optimized BKW algorithm solves this LWE instance with a total complexity of $2^{129.7}$, representing an improvement of 15.4 bits over the $2^{145.1}$ complexity of the coded-BKW algorithm implemented in lattice-estimator [4].

Lindner and Peikert [30] proposed new parameters for LWE-based cryptography. We consider values of n in the range $[128, 192, 320, 512]$. Table 3 presents a comparison of the time complexities of various BKW algorithms for solving LWE instances with Lindner-Peikert's parameters. As shown in Table 3, our algorithm outperforms the state-of-the-art coded-BKW in [4], achieving improvements in time complexity ranging from 9.1 to 24.4 bits. For $n = 192$, the coded-BKW algorithm in the lattice-estimator requires $2^{109.6}$ operations, while our algorithm reduces this to $2^{100.2}$, yielding a 9.1-bit improvement.

In Table 3, we also select some LWE parameters from the TU Darmstadt LWE challenge [1], with n taking values from $[40, 50, 70, 120]$. Our results show that our algorithm achieves an improvement in time complexity ranging from 3.3 to 17.1 bits compared to the commonly used coded-BKW algorithm in lattice-estimator [4].

Note 1. We provide a comprehensive overview of the total dimension reduction achieved by the *Code-reduce* technique across various LWE instances [1, 30, 37], including the corresponding code lengths, in Appendix A.

6.2 Memory Consumption Comparison

In Table 4, we compare the memory complexity of our optimized BKW algorithm with the state-of-the-art coded-BKW algorithm [23] as reported in the lattice-estimator [4].

As shown in Table 4, our BKW algorithm demonstrates a significant advantage in memory consumption compared to the coded-BKW algorithm [23] in the lattice-estimator [4]. For Regev's parameters [37] (excluding $n = 80$), our

Table 4: Comparison of estimated memory complexity (in $\log(\cdot)$) of the BKW algorithm for solving various LWE instances. The label ‘ [4]’ refers to the state-of-the-art coded-BKW algorithm [23] from lattice-estimator [4], while ‘Ours’ denotes the logarithmic memory complexity of our BKW algorithm. ‘LOG’ represents $\log m$ in n, and ‘**Impro**’ highlights the improvements of our results.

Regev Parameters [37]				
n	[4]	Ours	LOG	**Impro**
64	38.9	28.0	0.44	**10.9**
80	53.8	41.7	0.52	**12.1**
96	55.9	43.5	0.45	**12.4**
112	58.1	45.1	0.40	**13.1**
128	73.3	59.7	0.44	**13.7**
160	91.2	77.2	0.48	**14.0**
224	116.1	97.9	0.44	**18.2**
256	131.5	116.1	0.45	**15.4**
384	209.6	175.9	0.46	**33.7**
512	309.5	238.2	0.46	**71.3**
Lindner-Peikert Parameters [30]				
n	[4]	Ours	LOG	**Impro**
128	57.9	48.0	0.37	**9.9**
192	99.2	87.8	0.46	**11.4**
320	159.1	135.8	0.42	**23.3**

Lindner-Peikert Parameters [30]				
n	[4]	Ours	LOG	**Impro**
512	183.3	172.2	0.34	**11.1**
LWE Challenge Parameters [1]				
n	[4]	Ours	LOG	**Impro**
40	34.3	24.6	0.61	**9.7**
	34.3	24.5	0.61	**9.8**
	44.7	24.4	0.61	**20.3**
50	36.6	26.1	0.52	**10.5**
	47.5	36.6	0.73	**10.9**
	58.4	36.2	0.72	**22.2**
70	51.7	40.2	0.57	**11.5**
	63.8	40.0	0.57	**23.9**
	65.5	39.9	0.57	**25.6**
120	99.4	72.3	0.60	**27.1**
	73.3	58.7	0.49	**14.6**
	73.3	58.7	0.49	**14.7**

memory complexity remains below $2^{0.5n}$. Compared to the coded-BKW algorithm, our memory consumption shows improvements ranging from factors of $2^{10.9}$ to $2^{71.3}$. For instance, for $n = 512$, the memory consumption of our BKW algorithm is $m = 2^{238.2}$, which approximates $m \approx 2^{0.46n}$. This represents a factor of $2^{71.3}$ improvement over the memory consumption of the best coded-BKW algorithm [23] in the lattice-estimator [4].

For Lindner-Peikert's parameters [30], our memory consumption improvements range from factors of $2^{11.1}$ to $2^{23.3}$, with m falling within the range of $2^{0.34n}$ to $2^{0.46n}$. For the LWE parameters from the TU Darmstadt challenge [1], our memory consumption improvements range from factors of $2^{9.7}$ to $2^{27.1}$.

When memory is constrained to $2^{0.5n}$, we find that for 16 out of the 27 selected LWE instances, the actual memory consumption falls below $2^{0.5n}$. In summary, our improved BKW algorithm significantly outperforms the current best coded-BKW algorithm [4] in terms of memory complexity for the considered LWE instances.

6.3 BKW vs. Lattice Attacks

We provide a comparison between the state-of-the-art coded-BKW algorithm [23] in lattice-estimator [4], our optimized BKW algorithm, and the state-of-the-art hybrid dual attack [25] for Regev's parameters [37], as detailed in Table 5.

The hybrid dual attack demonstrates the best performance in both time and space complexity, particularly in high-dimensional scenarios. In contrast, our optimized BKW algorithm performs well in low-dimensional cases, such as $n = 80$ and $n = 96$. Although the BKW algorithm is generally less efficient than the hybrid dual attack, our optimizations narrow the performance gap between these two types of attacks.

Table 5: Comparison of time and memory complexities for coded-BKW [23], our optimized BKW, and hybrid dual attack [25].

	Time Complexity			Memory Complexity		
n	Coded-BKW	Ours	Hybrid dual	Coded-BKW	Ours	Hybrid dual
64	47.5	42.2	**41.4**	38.9	**28.0**	31.5
80	62.8	**53.4**	68.0	53.8	**41.7**	66.9
96	65.3	**55.5**	70.9	55.9	**43.5**	69.7
112	68.8	57.5	**55.6**	58.1	45.1	**44.7**
128	84.5	72.0	**60.8**	73.3	59.7	**57.3**
160	101.5	90.2	**84.4**	91.2	**77.2**	82.7
224	127.9	117.0	**93.8**	116.1	97.9	**83.7**
256	145.1	129.7	**105.4**	131.5	116.1	**101.2**
384	221.4	214.3	**156.2**	209.6	175.9	**153.3**
512	287.6	261.4	**201.9**	309.5	238.2	**195.6**

7 Conclusion

We optimize the BKW algorithm for LWE by designing a memory-efficient BKW algorithm that can automatically find the optimal reduction path. Our primary contributions lie in extending Bogos's work [9] to the LWE problem, introducing efficient memory-saving reduction techniques, and constructing a memory-optimized finite automaton. The proposed c-sum-PCS-reduce technique demonstrates lower time complexity compared to the state-of-the-art c-sum-dissect-reduce [17], particularly under a memory constraint to $2^{0.3n}$. For the selected LWE instances, our optimized BKW algorithm surpasses the state-of-the-art coded-BKW [23] in both time and memory complexities across all tested parameters.

In the future, several open challenges remain in refining the BKW algorithm for solving LWE. For instance, the independent heuristic used in the BKW algorithm requires experimental validation or theoretical proof. Additionally, developing general strategies that slightly trade runtime for better memory management will be crucial for future research. Finally, finding ways to adapt the BKW algorithm to stricter memory constraints is also a challenging problem. To adapt to stricter memory, the reduction phase can be partitioned into smaller subtasks, storing only the necessary blocks for each step, particularly in large-scale instances. Alternatively, sparse storage can be employed to record solely non-zero elements and their positions. Future work will validate whether these optimizations can effectively reduce the memory consumption of the BKW algorithm.

Acknowledgments We express our gratitude to the anonymous reviewers for their valuable feedback. This research was funded by the National Key Research and Development Program of China (Grant No. 2022YFB2702701) and the Key Research Program of the Chinese Academy of Sciences (Grant No. ZDRW-XX-2022-1).

References

1. TU Darmstadt LWE Challenge. https://www.latticechallenge.org/lwe_challenge/challenge.php, accessed: 2024-09-23
2. Albrecht, M.R., Cid, C., Faugère, J., et al.: On the complexity of the BKW algorithm on LWE. Des. Codes, Cryptogr. **74**, 325–354 (2015). https://doi.org/10.1007/s10623-013-9864-x
3. Albrecht, M.R., Faugère, J.C., Fitzpatrick, R., et al.: Lazy modulus switching for the BKW algorithm on LWE. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 429–445. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_25
4. Albrecht, M.R., Player, R., Scott, S.: Lattice estimator, https://github.com/malb/lattice-estimator, accessed: 2024-09-25
5. Applebaum, B., Cash, D., Peikert, C., et al.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.)

CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_35

6. Arora, S., Ge, R.: New algorithms for learning in presence of errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 403–415. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22006-7_34
7. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM **50**, 506–519 (2003). https://doi.org/10.1145/792538.792543
8. Bogos, S., Tramèr, F., Vaudenay, S.: On solving LPN using BKW and variants. Cryptogr. Commun. **8**, 331–369 (2016). https://doi.org/10.1007/s12095-015-0149-2
9. Bogos, S., Vaudenay, S.: Optimization of LPN solving algorithms. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 703–728. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_26
10. Budroni, A., Guo, Q., Johansson, T.: Improvements on making BKW practical for solving LWE. Cryptography **5**, 31 (2021). https://doi.org/10.3390/cryptography5040031
11. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_1
12. Delaplace, C., Esser, A., May, A.: Improved low-memory subset sum and LPN algorithms via multiple collisions. In: Albrecht, M. (ed.) IMACC 2019. LNCS, vol. 11929, pp. 178–199. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35199-1_9
13. Dinur, I., Dunkelman, O., Keller, N., et al.: Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 719–740 (2012). https://doi.org/10.1007/978-3-642-32009-542
14. Dinur, I.: An algorithmic framework for the generalized birthday problem. Des. Codes, Cryptogr. **87**(8), 1–30 (2018). https://doi.org/10.1007/s10623-018-0463-6
15. Dong, X., Hua, J., Sun, S., et al.: Meet-in-the-middle attacks revisited: Key-recovery, collision, and preimage attacks. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 278–308. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_10
16. Duc, A., Tramèr, F., Vaudenay, S.: Better algorithms for LWE and LWR. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 173–202. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_8
17. Esser, A., Heuer, F., Kübler, R., et al.: Dissection-BKW. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 638–666. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_22
18. Esser, A., Heuer, F., Kübler, R., et al.: Dissection-BKW. IACR Cryptol. ePrint Arch. **2018**, 569 (2018), https://eprint.iacr.org/2018/569
19. Fouque, P.A., Joux, A., Mavromati, C.: Multi-user collisions: Applications to discrete logarithm, even-mansour and prince. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 420–438. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_22
20. Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 1–20. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_1

21. Guo, Q., Johansson, T., Mårtensson, E., et al.: Coded-BKW with sieving. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 323–346. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_12
22. Guo, Q., Johansson, T., Mårtensson, E., et al.: On the asymptotics of solving the LWE problem using coded-BKW with sieving. IEEE Trans. Inf. Theory **65**, 5243–5259 (2019). https://doi.org/10.1109/TIT.2019.2906233
23. Guo, Q., Johansson, T., Stankovski, P.: Coded-BKW: Solving LWE using lattice codes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 23–42. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_2
24. Guo, Q., Mårtensson, E., Stankovski, P.: Modeling and simulating the sample complexity of solving LWE using BKW-style algorithms. Cryptogr. Commun. **15**, 331–350 (2023). https://doi.org/10.1007/s12095-022-00597-0
25. Guo, Q., Johansson, T.: Faster dual lattice attacks for solving LWE with applications to CRYSTALS. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13093, pp. 33–62. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92068-5_2
26. Herold, G., Kirshanova, E., May, A.: On the asymptotic complexity of solving lwe. Des. Codes, Cryptogr. **86**(1), 55–83 (2018). https://doi.org/10.1007/s10623-016-0326-0
27. Kirchner, P., Fouque, P.: An improved BKW algorithm for LWE with applications to cryptography and lattices. In: Gennaro, R. (ed.) CRYPTO 2015. LNCS, vol. 9215, pp. 43–62. Springer (2015). https://doi.org/10.1007/978-3-662-47989-6_3
28. Kirchner, P.: Improved generalized birthday attack. IACR Cryptol. ePrint Arch. **2011**, 377 (2011), https://api.semanticscholar.org/CorpusID:15196823
29. Levieil, E., Fouque, P.A.: An improved LPN algorithm. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 348–359. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11832072_24
30. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19074-2_21
31. Liu, H., Yu, Y.: A non-heuristic approach to time-space tradeoffs and optimizations for BKW. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. LNCS, vol. 13793, pp. 741–770. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22969-5_25
32. Liu, M., Nguyen, P.: Solving BDD by enumeration: An update. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 293–309. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36095-4_19
33. Micciancio, D., Regev, O.: Lattice-based cryptography. In: Bernstein, D., Buchmann, J., Dahmen, E. (eds.) Post-Quantum Cryptography, pp. 147–191. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-540-88702-7_5
34. Neyman, J., Pearson, E.: On the problem of the most efficient tests of statistical hypotheses. Philos. Trans. Roy. Soc. London Ser. A. **231**, 289–337 (1933). https://doi.org/10.1007/978-1-4612-0919-5_6
35. Nikolić, I., Sasaki, Y.: A new algorithm for the unbalanced meet-in-the-middle problem. In: Cheon, J., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 627–647. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_23
36. Qian, X., Liu, J., Gu, C., et al.: An improved BKW algorithm for LWE with binary uniform errors. In: ICCCS 2020. pp. 87–92. IEEE, Shanghai, China (2020). https://doi.org/10.1109/ICCCS49078.2020.9118492

37. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC 2005. pp. 84–93. New York, USA (2005). https://doi.org/10.1145/1060590.1060603
38. Schnorr, C.P., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math. Program. **66**, 181–199 (1994). https://doi.org/10.1007/BF01581144
39. Trimoska, M., Ionica, S., Dequen, G.: Time-memory trade-offs for parallel collision search algorithms. IACR Cryptol. ePrint Arch. (2017), https://eprint.iacr.org/2017/581
40. Van Oorschot, P., Wiener, M.: Parallel collision search with cryptanalytic applications. J. Cryptol. **12**(1), 1–28 (1999). https://doi.org/10.1007/s001459900030
41. Wiggers, T., Samardjiska, S.: Practically solving LPN. In: ISIT 2021. pp. 2399–2404 (2021). https://doi.org/10.1109/ISIT45174.2021.9518109
42. Zhang, X., Zheng, Z., Wang, X.: A detailed analysis of primal attack and its variants. Sci. China Inf. Sci. **65**, 132301 (2022). https://doi.org/10.1007/s11432-020-2958-9

A The Code-reduce technique

We adhere to the notation and analysis of coding noise within the *Code-reduce* technique, as introduced by Guo et al. [23]. Specifically, we assume that *Code-reduce* utilizes a concatenated code comprising s random linear lattice codes $[l_i, b]$ for $i = 1, 2, \ldots, s$, which reduces the dimension of sample vectors by $L_i = \sum_{i=1}^{s} l_i$.

We provide a comprehensive overview of the total dimension reduction achieved by the *Code-reduce* technique across various LWE instances, along with the corresponding code lengths employed in the process. The LWE parameters suggested by Regev [37], Lindner-Peikert [30], and the TU Darmstadt challenge [1] are detailed in Tables 6 to 8, respectively.

Table 6: Total dimension reduction and code lengths for Regev parameters [37] in our optimized BKW algorithm using *Code-reduce* technique.

Parameters	n	b	*Code-reduce*	l_i
Regev [37]	64	2	16	2,2,2,2,2,2,2,2
	80	3	14	4,4,3,3
	96	3	12	4,4,4
	112	3	24	4,4,4,3,3,3,3
	128	4	42	6,5,5,5,5,4,4,4,4
	160	5	33	7,7,7,6,6
	224	6	48	9,9,8,8,7,7
	256	7	101	11,10,10,9,9,8,8,8,7,7,7,7
	384	10	109	16,15,14,14,13,13,12,12
	512	13	229	21,20,19,18,18,17,16,16,15,15,14,14,13,13

Table 7: Total dimension reduction and code lengths for Lindner-Peikert parameters [30] in our optimized BKW algorithm using *Code-reduce* technique.

	n	b	*Code-reduce*	l_i
Lindner-Peikert [30]	128	4	22	5,5,4,4,4
	192	7	55	11,10,9,9,8,8
	320	11	110	17,16,15,14,13,12,12,11
	512	14	66	18,17,16,15

Table 8: Total dimension reduction and code lengths for LWE parameters from the TU Darmstadt challenge [1] in our optimized BKW algorithm using *Code-reduce* technique.

	n	q	σ	b	*Code-reduce*	l_i
LWE Challenge [1]	40	1601	8.01	2	16	3,3,2,2,2,2,2
		1601	16.01	2	16	3,3,3,3,2,2
		1601	24.02	2	17	4,4,3,3,3
	50	2503	12.52	2	23	3,3,3,2,2,2,2,2,2,2
		2503	37.55	3	32	7,6,5,5,5,4
		2503	75.09	3	38	8,7,7,6,5,5
	70	4903	24.52	3	33	5,5,5,4,4,4,3,3
		4903	49.03	3	39	6,6,5,5,5,4,4,4
		4903	73.55	3	46	7,6,6,5,5,5,4,4,4
	120	14401	72.01	5	78	10,10,9,8,8,7,7,7,6,6
		14401	144.01	4	71	10,9,8,8,7,7,6,6,5,5
		14401	216.02	4	77	11,10,9,8,8,7,7,6,6,5