A Non-heuristic Approach to Time-space Tradeoffs and Optimizations for BKW

Hanlin Liu[1] and Yu Yu[1,2]

[1] Shanghai Jiao Tong University, Shanghai 200240, China
[2] Shanghai Qi Zhi Institute, 701 Yunjin Road, Shanghai 200232, China
E-mail: hans1024@sjtu.edu.cn, yuyuathk@gmail.com

Abstract. Blum, Kalai and Wasserman (JACM 2003) gave the first sub-exponential algorithm to solve the Learning Parity with Noise (LPN) problem. In particular, consider the LPN problem with constant noise and dimension n. The BKW solves it with space complexity $2^{\frac{(1+\epsilon)n}{\log(n)}}$ and time/sample complexity $2^{\frac{(1+\epsilon)n}{\log(n)}} \cdot 2^{\Omega(n^{\frac{1}{1+\epsilon}})}$ for small constant $\epsilon \to 0^+$. We propose a variant of the BKW by tweaking Wagner's generalized birthday problem (Crypto 2002) and adapting the technique to a c-ary tree structure. In summary, our algorithm achieves the following:

1. **(Time-space tradeoff).** We obtain the same time-space tradeoffs for LPN and LWE as those given by Esser et al. (Crypto 2018), but without resorting to any heuristics. For any $2 \leq c \in \mathbb{N}$, our algorithm solves the LPN problem with time complexity $2^{\frac{\log(c)(1+\epsilon)n}{\log(n)}} \cdot 2^{\Omega(n^{\frac{1}{1+\epsilon}})}$ and space complexity $2^{\frac{\log(c)(1+\epsilon)n}{(c-1)\log(n)}}$ for $\epsilon \to 0^+$, where one can use Grover's quantum algorithm or Dinur et al.'s dissection technique (Crypto 2012) to further accelerate/optimize the time complexity.
2. **(Time/sample optimization).** A further adjusted variant of our algorithm solves the LPN problem with sample, time and space complexities all kept at $2^{\frac{(1+\epsilon)n}{\log(n)}}$, saving factor $2^{\Omega(n^{\frac{1}{1+\epsilon}})}$ for $\epsilon \to 0^+$ in time/sample compared to the original BKW, and the variant of Devadas et al. (TCC 2017).
3. **(Sample reduction).** Our algorithm provides an alternative to Lyubashevsky's BKW variant (RANDOM 2005) for LPN with a restricted amount of samples. In particular, given $Q = n^{1+\epsilon}$ (resp., $Q = 2^{n^\epsilon}$) samples for any constant $\epsilon > 0$, our algorithm saves a factor of $2^{\Omega(n)/\log(n)^{1-\kappa}}$ (resp., $2^{\Omega(n^\kappa)}$) for constant $\kappa \to 1^-$ in running time while consuming roughly the same space, compared with Lyubashevsky's algorithm.

In particular, the time/sample optimization benefits from a careful analysis of the error distribution among the correlated candidates, which was not studied by previous rigorous approaches such as the analysis of Minder and Sinclair (J.Cryptology 2012) or Devadas et al. (TCC 2017).

1 Introduction

1.1 The LPN problem and the BKW algorithm

The LPN problem with dimension $n \in \mathbb{N}$ and noise rate $0 < \mu < 1/2$ asks to recover the $\mathbf{s} \xleftarrow{\$} \mathbb{F}_2^n$ given an oracle that for each query responds with $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)$ for uniformly random $\mathbf{a}_i \xleftarrow{\$} \mathbb{F}_2^n$ and Bernoulli distributed error e_i, i.e., $\Pr[e_i = 1] = \mu$. Equivalently, LPN can be rephrased in the matrix-vector format, i.e., to recover $\mathbf{s}$ given $(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$, where $\mathbf{A}$ is a random $Q \times n$ Boolean matrix, $\mathbf{e} \leftarrow \mathcal{B}_\mu^Q$, '$\cdot$' and '$+$' denotes matrix vector multiplication and bitwise addition over $\mathbb{F}_2$. It is worth mentioning that a candidate solution can be verified with high confidence in polynomial time and space for any non-trivial noise rate $\mu \leq 1/2 - 1/\mathsf{poly}(n)$. A straightforward algorithm exhaustively searches for $\mathbf{s}$ (or any n-bit substring of $\mathbf{e}$ whose corresponding submatrix of $\mathbf{A}$ is invertible), which takes exponential time but consumes only polynomial-size space and thus can be applied in extreme space-constrained situations.

Blum, Kalai and Wassermann [6] gave the first sub-exponential algorithm (the BKW algorithm) that solves the LPN problem via an iterative block-wise Gaussian elimination method. Consider the $\mathsf{LPN}_{n,\mu}$ problem with dimension n, and noise rate $\mu = \frac{1-\gamma}{2}$. For block size b, and number of iterations a such that $ab = n$, the algorithm does the following (see Section 2.3 for more formal details):

1. Runs for a iterations and reduces the dimension by b bits in each iteration (by XORing LPN sample pairs whose corresponding block sum to zero). This results in samples in the form of $(\mathbf{u}_1, \langle \mathbf{u}_1, \mathbf{s} \rangle + \tilde{e}_j) = (\mathbf{u}_1, \mathbf{s}_1 + \tilde{e}_j)$, where $\mathbf{s}_1$ is the first bit of $\mathbf{s}$, and $\tilde{e}_j$ is the sum of noise from 2^a original LPN samples.
2. Repeats step 1 on fresh new LPN samples for $m \approx (1/\gamma)^{2^{a+1}}$ times, obtaining at least one candidate $(\mathbf{u}_1, \mathbf{s}_1 + \tilde{e}_j)$ each time.
3. Majority votes on the m samples obtained in step 2 and produces a candidate for $\mathbf{s}_1$. Repeats the process for other bits of $\mathbf{s}$ (on previously used samples).

The BKW solves the LPN problem in time T, using space of size M and up to Q samples, and succeeds with the probability P as below

$$T \approx 2^b \cdot (1/\gamma)^{2^{a+1}}, \;\; M \approx 2^b, \;\; Q \approx 2^b \cdot (1/\gamma)^{2^{a+1}}, \;\; P = 1 - \mathsf{negl}(n) \;\; ,$$

where throughout the paper "$\approx$" denotes the approximate relation that omits a multiplicative $\mathsf{poly}(n)$ factor. For any constant $0 < \gamma < 1$, we set $a = \frac{\log(n)}{1+\epsilon}$ and $b = \frac{(1+\epsilon)n}{\log(n)}$ such that $T \approx 2^{\frac{(1+\epsilon)n}{\log(n)}} \cdot 2^{\Omega(n^{\frac{1}{1+\epsilon}})}$ and $M \approx 2^{\frac{(1+\epsilon)n}{\log(n)}}$, where constant $\epsilon \to 0^+$. Quite naturally, one may raise the following questions:

1. **(Time-space tradeoff).** *Is it possible to achieve meaningful time-space tradeoffs for BKW to deal with bounded space in practice?*
2. **(Time/sample optimization).** *Is it possible to optimize the time/sample without sacrificing space, in particular, to eliminate the* $(1/\gamma)^{2^{a+1}}$ *factor?*

3. **(Sample reduction).** *Is it possible to push the sample complexity to a much lower order of magnitude than the time/space complexities?*

Below we first survey related works and progress made in tackling the above problems followed by a summary of our contributions.

1.2 Time-space Tradeoff for BKW

The huge space consumption of BKW has become an obstacle for a realistic security evaluation of LPN/LWE-based crypto-systems. As discussed in [18], while performing 2^{60} or more steps is considered doable with a reasonable budget, an algorithm consuming a space of size 2^{60} is definitely out of reach in practice. Likewise, in the lattice setting the enumeration method (e.g., the Kannan's algorithm [26] that takes time $2^{O(n\log(n))}$ and space $\mathsf{poly}(n)$) often beats the lattice sieving [16, 29–31] with time and space $2^{O(n)}$ in practice, and there is a renewed interest in the time-space trade-offs, e.g., lattice tuple sieving [4, 22, 23].

Esser et al. [18] discussed time-space tradeoff for BKW, but their algorithm already needs exponential time for space requirement below $2^{n/\log(n)}$. Later, they [17] introduced another variant of the BKW with better support for time-space trade-offs, called the c-sum BKW, where $2 \leq c \in \mathbb{N}$. Initially, it starts with a list of independent and uniformly random vectors $L_0 = (\mathbf{a}_{0,1}, \cdots, \mathbf{a}_{0,N})$, omitting the noisy parity bits for succinctness. It iteratively takes sums of c samples from the previous list L_i and stores those (that zero out the $(i+1)$-th b-bit block) into the next L_{i+1}, until at last it reaches a given target (typically of Hamming weight 1). The rest of the steps (repeating the above process m times, majority voting, etc.) are similar to the original BKW [6]. Note that c is the parameter to tune the tradeoff between space and time. In particular, $\binom{N}{c}$ increases exponentially with c, so with larger c one may use a smaller space at the cost of increasing time.

Nevertheless, the intermediate samples during each iteration of the c-sum BKW are somehow correlated, e.g., $\mathbf{a}_1+\mathbf{a}_2$, $\mathbf{a}_2+\mathbf{a}_3$ and $\mathbf{a}_1+\mathbf{a}_3$ are correlated in that they jointly sum to $\mathbf{0}$ regardless of the values of $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$. Note that the original BKW [6] resolves the independence issue by using 2^b reference vectors (whose i-th block take all values over $\mathbb{F}_2^b$) in each i-th iteration, and XORing the rest vectors with one of the reference vector (zeroing out the i-th block), which produces independent vectors for the next iteration. In the generalized c-sum setting [17], it is not clear how the independence can be guaranteed to obtain a rigorous analysis of the running time, space consumption and success rate. Esser et al. [17] resorted to the independence heuristic that simply assumes independence among those vectors, and they also provided some empirical evidences that the results (for certain parameter choices) behave close to the analysis under the idealized heuristics. We remark that similar independence heuristics were already used in the optimized analysis of concrete LPN instances (e.g., [7, 8, 20]).

Under the independence heuristics, Esser et al. [17] obtained various variants of the c-sum BKW, such as the naive c-sum BKW, dissection c-sum BKW, tailored dissection c-sum BKW, and quantum c-sum BKW, as shown in Table 1.

The naive c-sum BKW is the most generic one that admits time-space tradeoffs for arbitrary $2 \leq c \in \mathbb{N}$, the dissection c-sum BKW is the time-optimized version of the naive c-sum BKW for $c \in \{(i^2+3i+4)/2 : 0 \leq i \in \mathbb{N}\}$, the tailored dissection c-BKW is a fine-grained version of the dissection c-sum BKW (by adjusting the value of β, see also a visual illustration in [33, Figure 4]) that relies on additional heuristics, and the quantum c-sum BKW is the quantumly accelerated version of the naive c-sum BKW via the Grover algorithm [9, 15, 19]. They also applied the c-sum BKW to the LWE problem [36] and got similar results (see Table 5). We refer to Section 2.4 for more details about the c-sum BKW algorithm. Looking ahead, we provide unconditional algorithms with essentially the same complexities (see Section 3.2 through Section 3.7).

Table 1. The time and space complexities of the c-sum BKW [17] (and our c-sum$^+$ BKW) for solving the $\mathsf{LPN}_{n,\mu}$ problem, where $N_c = 2^{\frac{\log(c)}{c-1}\cdot\frac{n}{\log(n)}\cdot(1+\epsilon)}$ and constant $\epsilon > 0$.

	c-sum (c-sum$^+$) BKW	Space	Time	for
Classic	Original BKW [6]	N_2	N_2	$c = 2$
	Naive	N_c	N_c^{c-1}	$c \geq 2$
	Dissection	N_c	$N_c^{c-\sqrt{2c}}$	$c = 4, 7, 11, \cdots$
	Tailored Dissection	N_c^{β}	$N_c^{c-\beta\sqrt{2c}}$	$c = 4, 7, 11, \cdots \quad \beta \in [1, \frac{\sqrt{c}}{\sqrt{c}-1}]$
Quantum	Naive + Grover	N_c	$N_c^{c/2}$	$c \geq 2$

Table 2. A comparison of our time-space tradeoff and the heuristic state of the art [11, 13] for solving the $\mathsf{LPN}_{n,\mu}$ problem, where $N_c = 2^{\frac{\log(c)}{c-1}\cdot\frac{n}{\log(n)}\cdot(1+\epsilon)}$ and constant $\epsilon > 0$.

	Time-space tradeoff	for
Our dissection version	$T = N_c^{c-\sqrt{2c}} \quad M = N_c$	$c = 4, 7, 11, \cdots$
Dinur [13]	$T^{\log(n)/2+1} \cdot M^{\log(n)/2} = N_2^{\log(n)}$	$\sqrt{T} < M < T$
	$T^2 \cdot M^{3\log(n)/2-2} = N_2^{\log(n)}$	$M < \sqrt{T}$
Delaplace et al. [11]	$T = N_2^{(\frac{1}{2}+\frac{1}{c})\log(c)} \quad M = N_2^{\frac{2}{c}\log(c)}$	$c \geq 2$

We give the same end results as [17] while removing its underlying heuristics (see Table 1). We mention that [11, 13] further advanced the heuristic-based state-of-the-art with the aid of parallel collision search (PCS). PCS used the similar independence heuristics such as $H_{\mathbf{A}}(\mathbf{y}) = \mathbf{y}^T \cdot \mathbf{A}$ behaves like a random oracle or pseudorandom function, where $\mathbf{A}$ is the public matrix of the LPN problem. Note that the assumption doesn't hold in general even for secret $\mathbf{A}$, e.g., for $\mathbf{y}_1 = \mathbf{y}_2 + \mathbf{y}_3$ we have $H_{\mathbf{A}}(\mathbf{y}_1) = H_{\mathbf{A}}(\mathbf{y}_2) + H_{\mathbf{A}}(\mathbf{y}_3)$. As depicted in Table 2, it is quite challenging to do a comprehensive comparison with [11, 13].

For instance, when $\sqrt{T} < M < T$, Dinur [13] achieves roughly $T \cdot M \approx N_2^2$, e.g., $T \approx N_2^{4/3}$ and $M \approx N_2^{2/3}$. In contrast, our dissection version (see Theorem 6) achieves the same $T \approx N_2^{4/3}$ and $M \approx N_2^{2/3}$ by setting $c = 4$. Further, for $M < \sqrt{T}$ our result seems better than [13] (i.e., our $T^2 \cdot M^{3\log(n)/2-2} < N_2^{\log(n)}$), but the comparison is unfair as the explicit result analyzed/stated in [13] only considers $c = 4$ (generalizing to other c may yield different results). Delaplace et al. [11] is less superior to ours for $c < 11$, and it outperforms ours when $M < 2^{0.35n/\log(n)}$ ($c \geq 11$ in our case). Therefore, as far as time-space tradeoff is concerned, we mainly focus on [17], and leave it as future work on how to remove all the heuristics of [11, 13].

1.3 Time/sample Optimization and Sample Reduction for BKW

As discussed in Section 1.1, the BKW [6] repeats step 1 for $(1/\gamma)^{2^{a+1}} = 2^{\Omega(n^{\frac{1}{1+\epsilon}})}$ times and thus increases the time and sample complexities by the same factor. In fact, step 1 may have already produced sufficiently many samples $(\mathbf{s}_1 + \tilde{e}_j)$, and intuitively one just needs a majority vote to decode out $\mathbf{s}_1$. However, those noise, say $\tilde{e}_j$ and $\tilde{e}_{j'}$, are both the XOR sums of noise from the LPN samples, and they might not be (even pairwise) independent. Levieil and Fouque [32] used the LF1 technique to replace the majority voting and recover multiple secret bits (instead of a single one) at the same time. However, the BKW variant of [32] employs the LF2 technique when mixing up the vectors and heuristically assumes that the mixed up vectors behave as if they were independent, which is what we want to avoid in this paper. Devadas et al. [12] proposed a (non-heuristic) single-list pair-wise iterative collision search method to optimize the BKW, where they showed that the distribution of solutions is close to a Poisson distribution and applied the Chen-Stein method [3] of the second moment analysis to bound the difference. As a result, their variant solves the LPN problem (with overwhelming probability) in time T, using space of size M and sample complexity Q as below

$$T \approx 2^b \cdot (1/\gamma)^{2^a}, \quad M \approx 2^b \cdot (1/\gamma)^{2^a}, \quad Q \approx 2^b \ ,$$

where their sample complexity gets rid of the $(1/\gamma)^{2^{a+1}}$ factor as desired, time complexity is only mitigated (factor $(1/\gamma)^{2^{a+1}}$ squared to $(1/\gamma)^{2^a}$), and space complexity even deteriorates by factor $(1/\gamma)^{2^a}$ compared to the original BKW [6].

Lyubashevsky [34] studied how to solve the LPN problem with fewer samples. In particular, he used $Q = n^{1+\epsilon}$ (for constant $\epsilon > 0$) LPN samples as a basis to generate as many samples as needed, and feed them to the original BKW [6]. Concretely, let $(\mathbf{A}, \mathbf{t}^\mathsf{T} = (\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{x}^\mathsf{T}))$ be the initial LPN samples, where $\mathbf{A}$ is the $n \times Q$ matrix, and vectors with 'T' denote row vectors. A "re-randomized LPN" oracle take as input $(\mathbf{A}, \mathbf{t}^\mathsf{T})$ and responds with $(\mathbf{A}\mathbf{r}_i, \mathbf{t}^\mathsf{T}\mathbf{r}_i = \mathbf{s}^\mathsf{T}\mathbf{A}\mathbf{r}_i + \mathbf{x}^\mathsf{T}\mathbf{r}_i)$ as the i-th re-randomized LPN sample, where every $\mathbf{r}_i$ is drawn from the set of length-Q-weight-w strings uniformly at random. For an appropriate value of w, $(\mathbf{A}, \mathbf{A}\mathbf{r}_i, \mathbf{x}^\mathsf{T}\mathbf{r}_i)$ is statistically close to $(\mathbf{A}, \mathbf{U}_n, \mathbf{x}^\mathsf{T}\mathbf{r}_i)$ by the leftover hash lemma [25] with mildly strong noise $\mathbf{x}^\mathsf{T}\mathbf{r}_i$. In the end, Lyubashevsky's variant of

BKW solves the LPN problem (with overwhelming probability) in time T, using space of size M and sample complexity Q as below

$$T \approx 2^b \cdot (4/\gamma)^{2^{a+2} \cdot n/(\epsilon \log(n))}, \;\; M \approx 2^b, \;\; Q = n^{1+\epsilon} \;\; .$$

For constant $0 < \gamma < 1$, we set $a = \kappa \cdot \log\log(n)$ and $b = \frac{n}{\kappa \log\log(n)}$ for constant $0 < \kappa < 1$ and thus $T = 2^{\frac{n}{\kappa \log\log(n)}} \cdot 2^{\Omega(n)/\log(n)^{1-\kappa}}$, which is optimized when $\kappa \to 1^-$. Let us mention that Lyubashevsky's technique [34] also implies that LPN with $Q = 2^{n^\epsilon}$ (constant $0 < \epsilon < 1$) samples can be solved in time and space complexity $2^{O(n/\log(n))}$. We refer to Section 4.2 for more details.

1.4 Our Contributions

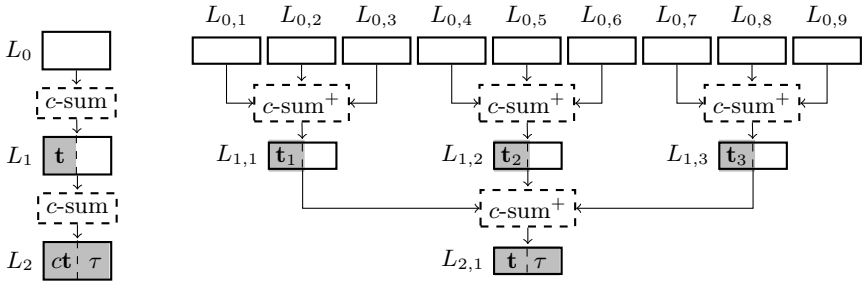
In this paper, we consider a problem that can be seen as a variant of Wagner's generalized birthday problem [38]. We recall the generalized birthday problem that, given $k = 2^a$ independent lists of i.i.d. uniformly random vectors, challenges to find out k vectors, one from each list, summing to a specified target, where the k vectors constitute a solution to the problem. The problem we consider further generalizes and differs to the generalized birthday problem in the following ways.

- (**Generalization**). We consider the case of $k = c^a$ for $2 \leq c \in \mathbb{N}$ and $a \in \mathbb{N}^+$.
- (**Pairwise-independence**). Each list consists of pairwise independent (instead of i.i.d. random) vectors, and all the lists are mutually independent.
- (**Bias analysis**). Our analysis framework extends to the case where each random vector is labelled with a true/false flag (to fully represent the LPN problem). We show that as long as the initial bias (the difference between the number of true and false samples) is bounded, the resulting bias among the solutions will be bounded (with reasonable blowup) as well, a feature not studied by the previous algorithms for the generalized birthday problem. [3]

As visualized in Fig. 1(b), our algorithm, referred to as the c-sum$^+$ BKW, breaks down the above problem on c^a lists into $(c^{a-1} + \cdots + c^0)$ subproblems on c lists, called the c-sum$^+$ problems. Further, we show that as long as the pairwise-independence (for vectors within each list) and mutual independence (among the lists) are satisfied for the c^a lists at the input level, the conditions will be satisfied by the lists at every other level (e.g., $L_{1,1}$, $L_{1,2}$, $L_{1,3}$ in Fig. 1(b)). We give analysis of the time, space and success probability without resorting to heuristics, thank to the pairwise-independence condition. Under our unified framework, the three tweaks, i.e., generalization, bias analysis and pairwise-independence, lead to the following advantages respectively.

1. **(Time-space tradeoff).** Our algorithm admits various time-space tradeoffs for solving LPN (shown in Table 1) and LWE (see Table 5), same as those achieved by the c-sum BKW [17], but without relying on any heuristiscs.

[3] The original generalized birthday problem omits LPN's noise labels. Even if many solutions are found, the correlations among the accumulated noise do not support majority voting. Therefore, non-heuristic analysis typically repeats the process on fresh new samples for $2^{n^{1-\epsilon}}$ times and thus incurs the same overhead on time/sample.



(a) c-sum BKW [17] (b) our c-sum$^+$ BKW (c=3, $\mathbf{t} = \mathbf{t}_1 + \mathbf{t}_2 + \mathbf{t}_3$)

Fig. 1. An illustration of the c-sum BKW [17] and our c-sum$^+$ BKW.

2. **(Time/sample optimization).** We carefully analyze and bound the error distribution of the correlated solutions in step 1 (e.g., $L_{2,1}$ in Fig. 1(b)), and therefore avoid the "repeat-m-times loop" in step 2. This saves a factor of $N_2 = (1/\gamma)^{2^{a+1}} = 2^{\Omega(n\frac{1}{1+\epsilon})}$ for small constant $\epsilon \to 0^+$ in time and sample complexities compared to the original BKW [6]. Our algorithm also enjoys a sub-exponential $\sqrt{N_2}$ advantage in time and space complexities compared to the optimized BKW of Devadas et al. [12]. See Table 3 for more details.
3. **(Sample reduction).** By using pairwise independent samples for the initial lists, we provide an alternative to Lyubashevsky's BKW variant [34] with improved time complexity. In particular, given $Q = n^{1+\epsilon}$ (resp., $Q = 2^{n^\epsilon}$) samples for constant $\epsilon > 0$, our algorithm saves a factor of $2^{\Omega(n)/\log(n)^{1-\kappa}}$ (resp., $2^{\Omega(n^\kappa)}$) for constant $\kappa \to 1^-$ in running time compared with the counterpart in [34]. We refer to Table 4 and Section 4.2 for details.

Table 3. The space, time and sample complexities of different variants of the BKW for solving the $\mathsf{LPN}_{n,\mu}$ problem with $\mu = (1-\gamma)/2$, under condition $N_1 \approx N_2$, where $ab = n$, $N_1 = 2^b$ and $N_2 = (1/\gamma)^{2^{a+1}}$ disregarding $\mathsf{poly}(n)$ factors.

Algorithm	Space	Time	Sample	Condition
The original BKW [6]	N_1	$N_1 \cdot N_2$	$N_1 \cdot N_2$	$N_1 \approx N_2$
Devadas et al.'s [12]	$N_1 \cdot \sqrt{N_2}$	$N_1 \cdot \sqrt{N_2}$	N_1	$N_1 \approx N_2$
Ours	N_1	N_1	N_1	$N_1 \approx N_2$

It might seem counter-intuitive that our results listed in Table 3 and Table 4 only depend on N_1 but still needs to satisfy $N_1 \approx N_2$ (or similar ones in Table 4) for optimized time complexity. As we will see, the condition $N_1 \geq N_2$ (or alike) is translated from the condition that sufficient amount of samples are needed to ensure the correctness of majority voting (see Theorem 9), and we thus let $N_1 \approx N_2$ for optimized complexity and fair comparison.

Table 4. The space, time and sample complexities of different variants of the BKW for solving the $\mathsf{LPN}_{n,\mu}$ problem with $\mu = (1-\gamma)/2$, where $ab = n$, $N_1 = 2^b$, $N_2 = (4/\gamma)^{2^{a+2}\cdot n/(\epsilon\log(n))}$ and $N_2' = (4/\gamma)^{2^{a+2}\cdot n^{1-\epsilon}}$ and constant $\epsilon > 0$.

Sample	Algorithm	Space	Time	Condition
$n^{1+\epsilon}$	Lyubashevsky's [34]	N_1	$N_1 \cdot N_2$	$N_1 \approx N_2$
	Ours	N_1	N_1	$(N_1)^{\log\log(n)} \approx N_2$
2^{n^ϵ}	Lyubashevsky's [34]	N_1	$N_1 \cdot N_2'$	$N_1 \approx N_2'$
	Ours	N_1	N_1	$(N_1)^{\log(n)} \approx N_2'$

Related work. Minder and Sinclair [35] used second-moment analysis, and gave rigorous time/space bounds of Wagner's generalized problem. While it looks promising that the analysis of Minder and Sinclair [35] can be adapted to our further generalized case of $c \geq 2$ and pairwise-independent vectors (within each list), their approach does not support bias analysis. Recall that the original generalized birthday problem omits the noise labels, $e_1, \cdots, e_N$, from the LPN. Therefore, even many k-sum solutions are found, the correlations among the accumulated noise may not support majority voting. Note that even the pairwise independence condition does not hold for the noise, e.g., e_1, e_2, $e_1 + e_2$ are pairwise independent only for uniformly random (not for biased) e_1 and e_2. This is why previous non-heuristic algorithms have to repeat the process on fresh new samples for $N_2 = 2^{n^{1-\epsilon}}$ times and thus incurs the same overhead on time/sample. Recently, Devadas et al. [12] partially mitigated the issue by bounding the voting difference using the Chen-Stein method [3], but their bound is not as good as ours. As shown in Table 3, our result removes this penalty factor $m = N_2$ almost for free (without significantly increasing the time/sample complexity).

2 Preliminary

2.1 Notation

We use $\log(\cdot)$ to denote the binary logarithm. For $a \leq b \in \mathbb{N}$, $[a,b] \stackrel{\mathsf{def}}{=} \{a, a+1, \cdots, b\}$ and $[a] := [1,a]$. $|\mathcal{S}|$ is the cardinality of the set $\mathcal{S}$. For any set $\mathcal{S}$ and $0 \leq s \leq |\mathcal{S}|$, $\binom{\mathcal{S}}{s}$ denotes the set of all size-s subsets of $\mathcal{S}$. A list $L = (l_1, \cdots, l_N)$ is an element from set $\mathcal{S}^N$ with length $|L| = N$. We denote the empty list by $\emptyset$.

For $\mathbf{x} \in \mathbb{F}_2^n$ and $b < n$ we denote the last b coordinates of $\mathbf{x}$ by $\mathsf{low}_b(\mathbf{x})$. $\mathbf{u}_i$ denotes the i-th unit vector, and $\mathbf{0}^b$ denotes the zero vector of dimension b. We use ' :=' to denote deterministic value assignment. $\mathcal{U}_\mathcal{S}$ denotes the uniform distribution over set $\mathcal{S}$. $\mathcal{B}_\mu$ denotes the Bernoulli distribution with parameter μ, i.e., for $x \leftarrow \mathcal{B}_\mu$ we have $\Pr[x=1] = \mu$ and $\Pr[x=0] = 1-\mu$. We use $\mathbf{s} \stackrel{\$}{\leftarrow} \mathcal{S}$ (resp., $\mathbf{s} \leftarrow S$) to denote sampling $\mathbf{s}$ from set $\mathcal{S}$ uniformly at random (resp., according to distribution S). For $L = (l_1, \cdots, l_N)$ with every l_i uniformly distributed over $\mathbb{F}_2^b$, we say that L consists of pairwise independent elements if for every $1 \leq i < j \leq N$ the corresponding (l_i, l_j) is uniform over $\mathbb{F}_2^{2b}$.

Lemma 1 (Piling-up Lemma). *For $0 < \mu < 1/2$ and random variables e_1, e_2, $\cdots$, e_ℓ that are i.i.d. to $\mathcal{B}_\mu$ we have* $\Pr[\bigoplus_{i=1}^{\ell} e_i = 1] = \frac{1}{2}(1-(1-2\mu)^\ell)$.

Lemma 2 (Chebyshev's Inequality). *Let X be any random variable (taking real number values) with expectation μ and standard deviation σ (i.e., $Var[X] = \sigma^2 = \mathbb{E}[(X-\mu)^2]$). Then, for any $\delta > 0$ we have* $\Pr\left[|X-\mu| \geq \delta\sigma\right] \leq \frac{1}{\delta^2}$.

Lemma 3. *For pairwise independent real-valued r.v.s $X_1, \cdots, X_m$ it holds that*

$$Var\Big[\sum_{i=1}^{m} X_i\Big] = \sum_{i=1}^{m} Var\Big[X_i\Big] \ .$$

We defer the proof of Lemma 3 to the full version of the paper [33].

2.2 The Learning Parity with Noise Problem

The LPN problem comes with two versions, the decisional LPN and the search LPN, which are polynomially equivalent [2, 5, 27]. Therefore, we only state the search version for simplicity.

Definition 1 (Learning Parity with Noise) *For $n \in \mathbb{N}$, $\mathbf{s} \in \mathbb{F}_2^n$ and $0 < \mu < 1/2$, denote by* Sample *an oracle that, when queried, picks $\mathbf{a} \xleftarrow{\$} \mathbb{F}_2^n$, $e \leftarrow \mathcal{B}_\mu$ and outputs a sample of the form $(\mathbf{a}, l = \langle \mathbf{a}, \mathbf{s}\rangle + e)$. The* $\mathsf{LPN}_{n,\mu}$ *problem refers to recovering the random secret* [4] *$\mathbf{s}$ given access to* Sample. *We call n the dimension, $\mathbf{s}$ the secret, μ the error rate, l the label of $\mathbf{a}$ and e the noise.*

2.3 The Original BKW

The BKW algorithm [6] works in iterations, and during each i-th iteration, it uses 2^b reference vectors (whose i-th block take all values over $\mathbb{F}_2^b$). The rest vectors are added with the corresponding reference vector to zero out the i-th block, which yields new labels with doubled noise (the sum of a reference vector and another) and losing 2^b vectors in each iteration. The procedure repeats for b iterations (i.e., zeros out ab bits) until reaching a unit vector, say $\mathbf{u}_1$, and let the corresponding label be a candidate for $\langle \mathbf{u}_1, \mathbf{s}\rangle = \mathbf{s}_1$. One further repeats the above on new samples and does a majority vote to recover $\mathbf{s}_1$ with overwhelming probability. The procedure to recover other bits of $\mathbf{s}$ is likewise.

Theorem 1 (The BKW algorithm [6]) *For dimension n, block size b and number of blocks a such that $ab \geq n$, there is an algorithm that succeeds (with an overwhelming probability) in solving the* $\mathsf{LPN}_{n,\mu}$ *problem in time $T \approx 2^b \cdot (1/\gamma)^{2^{a+1}}$ and using space of size $M \approx 2^b$, where the noise rate $\mu = 1/2 - \gamma/2$.*

Concretely, for constant $0 < \epsilon < 1$, we set $a = \frac{\log(n)}{1+\epsilon}$ and $b = \frac{(1+\epsilon)n}{\log(n)}$ such that T and M are both on the order of $2^{\frac{(1+\epsilon)n}{\log(n)} + O(1)n^{\frac{1}{1+\epsilon}}} \approx 2^{\frac{(1+\epsilon+o(1))n}{\log(n)}}$.

[4] The distribution of the secret is typically uniform over $\mathbb{F}_2^n$, but it has no effect on the complexity of the BKW-style algorithms and thus is irrelevant in our context.

2.4 The c-sum Problem and c-sum BKW

Given a list of N (typically uniformly random) vectors, the c-sum problem challenges to find out c of them whose (XOR) sum equals a specified target (typically $\mathbf{0}^b$). Esser et al. [17] considered the variant that aims to find sufficiently many (at least N) such solutions. Notice that N is both the number of vectors in the input list and the amount of solutions produced as output. As we will later see, this (together with the independence heuristics) enables the c-sum BKW algorithm [17] to work from one iteration to another without losing samples.

Definition 2 (The c-sum Problem (c-SP) [17]) *Let $b, c, N \in \mathbb{N}$ with $c \geq 2$. Let $L \stackrel{\mathsf{def}}{=} (\mathbf{a}_1, \cdots, \mathbf{a}_N)$ be a list where $\mathbf{a}_i \stackrel{\$}{\leftarrow} \mathbb{F}_2^b$ independently for all i and let $\mathbf{t} \in \mathbb{F}_2^b$ be a target. A single-solution of the c-sum problem is a size-c set $\mathcal{L} \in \binom{[N]}{c}$ such that $\bigoplus_{j\in\mathcal{L}} \mathbf{a}_j = \mathbf{t}$. A complete-solution is a set of at least N distinct single-solutions.*

Esser et al. [17] proposed a variant of the BKW, referred to as the c-sum BKW, that admits time-space tradeoffs. This is achieved by generalizing the original BKW [6], which zeroes out one block per iteration by taking the sum of two vectors (i.e., 2-sum), to one that generates new samples that are the sum of c samples from previous iterations for arbitrary $2 \leq c \in \mathbb{N}$. It turns out that the c-sum BKW algorithm significantly reduces the space needed, as $\binom{N}{c}$ blows up exponentially with respect to c, at the cost of increased running time.

Algorithm 1: The c-sum BKW

Input: access to the oracle $\mathsf{LPN}_{n,\mu}$
Output: $\mathbf{s} \in \mathbb{F}_2^n$

1 $a := \frac{\log(n)}{(1+\epsilon_a)\log(c)}$, $b := \frac{n}{a}$, $m := \frac{8(1-\mu)n}{(1-2\mu)^{2c^a}}$, $N := 2^{\frac{b+c\log(c)+1}{c-1}}$;
2 **for** $i \leftarrow 1, \cdots, m$ **do**
3 Get N fresh LPN samples and save them in L;
4 **for** $j \leftarrow 1, \cdots, a-1$ **do**
5 $L \leftarrow c\text{-sum}(L, j, 0^b)$;
6 $L \leftarrow c\text{-sum}(L, a, \mathbf{u}_1)$;
7 **if** $L = \emptyset$ **then**
8 **Return** $\perp$;
9 Pick $(\mathbf{u}_1, b_i)$ uniformly from L;
10 $\mathbf{s}_1 \leftarrow \text{majorityvote}(b_1, \cdots, b_m)$;
11 Determine $\mathbf{s}_2, \cdots, \mathbf{s}_n$ the same way;
12 **Return** $\mathbf{s} = \mathbf{s}_1 \ldots \mathbf{s}_n$;

We recall the c-sum BKW in Algorithm 1. For a block size b and $j \in [a]$, let the coordinates $[n - jb + 1, n - (j-1)b]$ denote the j-th stripe. The important component of the c-sum BKW algorithm is the c-sum algorithm (see line 5 and

6) that generates some refresh samples whose j-th stripe for $j \in [a-1]$ (resp. the a-th stripe) is zeros (resp. the first unit vector). If the above steps generate some label-$\mathbf{u}_1$ samples, we pick one of these $(\mathbf{u}_1, b_i)$ sample uniformly at random (see line 9). Determining the first bit $\mathbf{s}_1$ with overwhelming probability needs sufficiently many independent label-$\mathbf{u}_1$ samples via the for-loop (see line 2). The process of recovering other bits $\mathbf{s}_i$ is likewise (by reusing the LPN samples).

Independence Heuristic [17]. However, the output samples of the c-sum algorithm are somehow correlated and may not feed into the next c-sum algorithm, which requires independent samples for its input (see Definition 2). For instance, the output of a 2-sum algorithm $\mathbf{a}_1+\mathbf{a}_2$, $\mathbf{a}_2+\mathbf{a}_3$ and $\mathbf{a}_1+\mathbf{a}_3$ are correlated in the sense that they sum to $\mathbf{0}$ regardless of the values of $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$. Esser et al. [17] introduced the independence heuristic that assumes independence among those vectors. Similar independence heuristics were already used in the optimized analysis of concrete LPN instances [7, 8, 20].

3 The c-sum$^+$ BKW and Time-space tradeoffs

In this section, we introduce the k-Generalized Birthday Problem [38], and breaks it down into many instances of c-sum$^+$ problems, where $k = c^a$. By giving solutions, optimizations, and speedups to the c-sum$^+$ problems, we get many variants of BKW algorithm (referred to as the c-sum$^+$ BKW), which achieve the same complexities (up to polynomial factors) as the counterparts of c-sum BKW by Esser et al. [17] without relying on heuristics.

We consider the k-Generalized Birthday Problem: there are $k = c^a$ lists $L_{0,1}$, …, L_{0,c^a}, where each $L_{0,i} \stackrel{\mathsf{def}}{=} (\mathbf{a}_{i,1}, \cdots, \mathbf{a}_{i,N})$ has N vectors, and satisfies

1. **(Intra-list pairwise independence)**. Within each list $L_{0,i}$, each $\mathbf{a}_{i,j}$ is uniformly random, and every pair of distinct vectors is pairwise independent, i.e., for all $j \neq k$ $(\mathbf{a}_{i,j}, \mathbf{a}_{i,k})$ is uniformly random.
2. **(Inter-list independence)**. $L_{0,1}, \cdots, L_{0,c^a}$, each seen as a random variable, are all mutually independent.

A solution of the problem is to find k vectors, one from each list, that sum to a specified target $\mathbf{t}$, i.e., $(j_1, \cdots, j_k) \in [N]^k$ such that $\bigoplus_{i=1}^{k} \mathbf{a}_{i,j_i} = \mathbf{t}$. The goal of the problem is to find as many (N or more) solutions as possible.

Further, in the extended k-Generalized Birthday Problem, we associate lists $E_{0,1}$, …, E_{0,c^a} with $L_{0,1}$, …, L_{0,c^a} respectively, where $E_{0,i} \stackrel{\mathsf{def}}{=} (\mathbf{e}_{i,1}, \cdots, \mathbf{e}_{i,N}) \in \mathbb{F}_2^N$ is the list of noise labels, and define the noise label of a solution as $\bigoplus_{i=1}^{k} \mathbf{e}_{i,j_i}$ accordingly. In addition to finding out N or more solutions, the extended problem also requires the noise labels of the solutions are biased (i.e., more 0-labels than 1-labels) as long as the noise of each $L_{0,1}$, …, L_{0,c^a} is sufficiently biased. Since this subsection serves to remove the heuristics of (and gives results fully comparable to) [17], we defer the extended problem and contributions of optimized time/sample optimization to Section 4.

3.1 The c-sum$^+$ Problem

Before presenting our c-sum$^+$ BKW, we first define the c-sum$^+$ problem below. Unlike the c-sum problem (see Definition 2) that produces c-sums from a single list, the c-sum$^+$ problem takes as input c lists and asks to find c vectors, one from each list, that sum to a given target. Furthermore, we require that the c lists are mutually independent, each consisting of pairwise independent vectors.

Definition 3 (The c-sum$^+$ Problem (c-SP$^+$)) *Let $b, c, N \in \mathbb{N}$ with $c \geq 2$. Let $L_1, \cdots, L_c$, $L_i \stackrel{\mathsf{def}}{=} (\mathbf{a}_{i,1}, \cdots, \mathbf{a}_{i,N}) \in \mathbb{F}_2^{b \cdot N}$, satisfy the Intra-list pairwise independence and Inter-list independence conditions (as defined in the k-Generalized Birthday Problem).*

Further, let $\mathbf{t} \in \mathbb{F}_2^b$ be a target. A solution of the c-sum$^+$ problem is a size-c list $K \stackrel{\mathsf{def}}{=} (k_1, \cdots, k_c) \in [N]^c$ such that $\bigoplus_{i=1}^{c} \mathbf{a}_{i,k_i} = \mathbf{t}$.

In fact, we will need the c-sum$^+$ problem to give at least N solutions (instead of a single one) in order to form another list for the subsequent iterations in our BKW algorithm. As stated in the lemma below, the pairwise independence already ensures the existence of sufficiently many (i.e., N) solutions albeit with less strong error probability, i.e., $2/N$ instead of $2^{-\Omega(N)}$ assumed under the independence heuristic [17]. As we will see, $2/N = \mathsf{negl}(n)$ for a super-polynomial N already suffices.

Lemma 4. *For $N = 2^{\frac{b+1}{c-1}}$, the c-SP$^+$ problem (as per Definition 3) has at least N and at most $3N$ solutions with the probability more than $1 - 2/N$.*

Proof. For every $K = (k_1, \cdots, k_c) \in [N]^c$ define a $0/1$-valued variable X_K that takes value $X_K = 1$ iff $\bigoplus_{i=1}^{c} \mathbf{a}_{i,k_i} = \mathbf{t}$. Thus, $X = \sum_K X_K$ is the number of solutions to the c-sum$^+$ problem, where every $K \in [N]^c$ has expectation $\mathbb{E}[X_K] = 2^{-b}$ and all the X_K are pairwise independent. Therefore,

$$\Pr\Big[|X - 2N| > N\Big] \leq \Pr\Big[|X - \mathbb{E}[X]| > N\Big] \leq \frac{Var[X]}{N^2} = \frac{\sum_S Var[X_S]}{N^2} \leq = \frac{2}{N} \ ,$$

where the first inequality is due to $N^{c-1} = 2^{b+1}$ and $\mathbb{E}[X] = N^c \cdot 2^{-b} = 2N$, and the second inequality is based on Chebyshev's inequality, the first equality is due to Lemma 3, and the last inequality is due to $Var[X_i] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 \leq \mathbb{E}[X_i^2] = \mathbb{E}[X_i]$. □

3.2 The c-sum$^+$ BKW

We introduced the c-sum$^+$ problem in Definition 3, and we show in Lemma 4 that it has at least N solutions (except with the probability $2/N$). We defer the concrete algorithms (and optimizations) for finding out the N solutions to a later stage. Instead, we assume a solver for c-sum$^+$ with time $T_{c,N,b}$ and space $M_{c,N,b}$, and then show how our c-sum$^+$ BKW algorithm breaks down the LPN problem into many instances of the c-sum$^+$ problem.

Abstractly speaking, our c-sum$^+$ BKW algorithm employs a c-ary tree of depth a (see Fig. 2 for an illustration of $a = 2$, $c = 3$), where each node represents a list of vectors, and each parent-node list consists of vectors each of which is the sum of c vectors from its c child nodes respectively (one from each child node). Further, we assume that for every parent node list $\left\{ \bigoplus_{i=1}^{c} \mathbf{a}_{i,k_i} \middle| (k_1, \cdots, k_c) \in [N]^c \right\}$ the choices $(k_1, \cdots, k_c)$ of the c-sums are independent of the values of its child lists L_1, ..., L_c, where $L_i = (\mathbf{a}_{i,1}, \cdots, \mathbf{a}_{i,N})$. While this independence assumption may seem contradictory to the c-sum$^+$ problem that seeks solutions satisfying $\bigoplus_{i=1}^{c} \mathbf{a}_{i,k_i} = \mathbf{t}$, we stress that this is due to the simplification of the problem. That is, our c-sum$^+$ BKW algorithm, just like the original BKW [6], zeros out the coordinates in iterations: at the j-the iteration, it finds the linear combinations of the j-th stripes that sum to zero, and produces the same combinations of the $(j+1)$-th stripes as the resulting list for the next iteration, i.e., $\left\{ \bigoplus_{i=1}^{c} \mathbf{a}_{i,k_i}^{j+1} \middle| (k_1, \cdots, k_c) \in [N]^c, \bigoplus_{i=1}^{c} \mathbf{a}_{i,k_i}^{j} = \mathbf{t} \right\}$, where the choice $(k_1, \cdots, k_c)$ is independent of the set of $(j+1)$-th stripe vectors $\{\mathbf{a}_{i,k}^{j+1} | i \in [c], k \in [N]\}$ to be combined.

Under the above simplified model, we have the following lemma that states that the leaf-level lists satisfy the intra-list pairwise independence and inter-list independence conditions (see Definition 3), then the conditions will preserved and propagated to all the non-leaf list nodes, all the way down to the root.

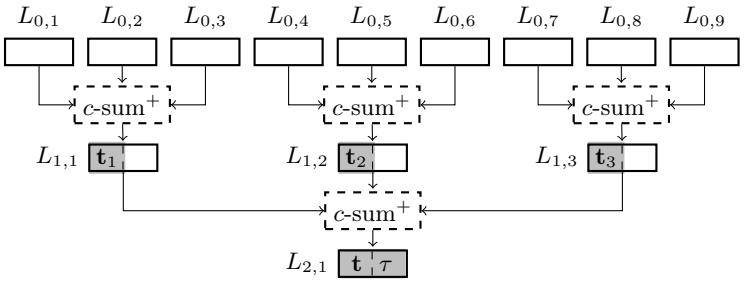


Fig. 2. An illustration of the c-sum$^+$ BKW for $c = 3$, where $\mathbf{t} = \mathbf{t}_1 + \mathbf{t}_2 + \mathbf{t}_3$

Lemma 5 (Pairwise independence preserving). *If the leaf-level lists $L_{0,1}$, ..., L_{0,c^a} are all mutually independent, and each $L_{0,i}$ consists of pairwise independent vectors. Then, for every $1 \le j \le a$ it holds that $L_{j,1}$, ..., $L_{j,c^{a-j}}$ are mutually independent, and every $L_{j,i}$ (for $1 \le i \le c^{a-j}$) consists of pairwise independent vectors.*

Proof. The proof follows by induction, namely, if the condition holds for level j, then it is also true for level $j + 1$. The mutual independence follows from the tree structure, i.e., if $L_{j,1}$, ..., $L_{j,c^{a-j}}$ are all mutually independent, then

so are the next-level parents $L_{j+1,1}, \ldots, L_{j+1,c^{a-j-1}}$ since each parent only depends on its own children nodes. Moreover, if at level j, $L_{j,1}, \ldots, L_{j,c^{a-j}}$ are all mutually independent and every list $L_{j,i}$ (for $1 \le i \le c^{a-j}$) consists of pairwise independent vectors, then at level $j+1$ we need to show that every list $L_{j+1,i'}$ (for $i' \in [c^{a-j-1}]$) consists of pairwise independent vectors as well. Consider any two vectors from $L_{j+1,i'}$ that are distinct c-sums of its child lists, say $\bigoplus_{\ell=1}^{c} \mathbf{a}_{\ell,k_\ell}$ and $\bigoplus_{\ell=1}^{c} \mathbf{a}_{\ell,k'_\ell}$. Then, there exists at least one $\ell \in [c]$ such that $k_\ell \neq k'_\ell$ and $(\mathbf{a}_{\ell,k_\ell}, \mathbf{a}_{\ell,k'_\ell}) \sim \mathcal{U}_{\mathbb{F}_2^{2b}}$ (as they are from the same list at level j which has pairwise independent vectors) and they are independent from other summand vectors in the c-sum (since the lists at level j are all mutually independent). It follows that $(\bigoplus_{\ell=1}^{c} \mathbf{a}_{\ell,k_\ell}, \bigoplus_{\ell=1}^{c} \mathbf{a}_{\ell,k'_\ell})$ is jointly uniform over $\mathbb{F}_2^{2b}$ and thus are pairwise independent. □

We can now reduce the problem of solving LPN to (many instances of) the c-sum$^+$ problem without relying on any heuristics (thanks to the pairwise independence preserving property by Lemma 5). The algorithm is formally described in Algorithm 2. For a block size b and $j \in [a]$, let the coordinates $[n - jb + 1, n - (j-1)b]$ denote the j-th stripe. Our algorithm proceeds level by level. At the 0-th level, the algorithm gets fresh LPN sample to initialize every list $L_{0,k}$ for $k \in [c^a]$ with $|L_{0,k}| = N = 2^{\frac{b+1}{c-1}}$ (see line 1). Then, at each j-th level ($1 \le j \le a-1$) our algorithm invokes c-sum$^+$ that takes as input the lists $L_{j-1,c(k-1)+1}, \cdots L_{j-1,ck}$ at the $(j-1)$-th level, and produces as output list $L_{j,k}$ at the j-th level (see lines $4-6$). The execution on the a-th (root) level is slightly different, i.e., we only need to solve a single instance of the c-sum$^+$ with target $\mathbf{u}_1$ (instead of zero), and produces a single solution (instead of N solutions). In other words, the code at line 10 is somewhat unnecessary in that it first produces N solutions (stored in $L_{a,1}$) but only (randomly) picks one of them, which is another problem we are going to tackle in the next section. Finally, we repeat the above many times on fresh LPN samples, and majority vote to decode out first secret bit. The recovery of other secret bits is likewise (reusing the samples). The c-sum$^+$ algorithm is an important building block of the c-sum$^+$ BKW. We state below their relations in terms of correctness and complexity.

Theorem 2 (The c-sum$^+$ BKW) *The $\mathsf{LPN}_{n,\mu}$ problem with $\mu = 1/2 - \gamma/2$ can be solved in time T and space M with the probability P as below*

$$T \approx T_{c,N,b} \cdot c^a \cdot (\frac{1}{\gamma})^{2 \cdot c^a},\ M \approx M_{c,N,b} \cdot c^a,\ P \ge 1 - \frac{1}{N} \cdot c^a \cdot (\frac{1}{\gamma})^{2 \cdot c^a} \cdot \mathsf{poly}(n) - \frac{n}{2^n},$$

where $T_{c,N,b}$ and $M_{c,N,b}$ are respectively the time and space complexities of the c-sum$^+$ algorithm that aims for N distinct solutions to the c-sum$^+$ problem with block (target) size b, $ab \ge n$, and $N = 2^{\frac{b+1}{c-1}}$ for $2 \le c \in \mathbb{N}$.

Notice: for now we omit the sample complexity since $Q \approx T$ under the scenario of unlimited samples, as opposed to the setting considered in Section 4.2.

Proof. The c-sum$^+$ algorithm is used to instantiate the c-sum$^+$ subroutine in Algorithm 2. As discussed in Lemma 4, the c-sum$^+$ problem (implicitly defined

Algorithm 2: The c-sum$^+$ BKW

Input: access to the oracle $\mathsf{LPN}_{n,\mu}$
Output: $\mathbf{s} \in \mathbb{F}_2^n$

1 $a := \frac{\log(n)}{(1+\epsilon_a)\log(c)}$, $b := \frac{n}{a}$, $m := \frac{8(1-\mu)n}{(1-2\mu)^{2c^a}}$, $N := 2^{\frac{b+1}{c-1}}$;
2 **for** $i \leftarrow 1, \cdots, m$ **do**
3 Save fresh LPN samples in $L_{0,1}$, ..., L_{0,c^a}, each of size N;
4 **for** $j \leftarrow 1, \cdots, a-1$ **do**
5 **for** $k \leftarrow 1, \cdots, c^{a-j}$ **do**
6 $L_{j,k} \leftarrow c\text{-sum}^+(L_{j-1,c(k-1)+1}, \cdots, L_{j-1,ck}, j, 0^b)$;
7 $L_{a,1} \leftarrow c\text{-sum}^+(L_{a-1,1}, \cdots, L_{a-1,c}, a, \mathbf{u}_1)$;
8 **if** $L_{a,1} = \emptyset$ **then**
9 **Return** $\perp$;
10 Pick $(\mathbf{u}_1, b_i)$ uniformly from $L_{a,1}$;
11 $\mathbf{s}_1 \leftarrow \text{majorityvote}(b_1, \cdots, b_m)$;
12 Determine $\mathbf{s}_2, \cdots, \mathbf{s}_n$ the same way;
13 **Return** $\mathbf{s} = \mathbf{s}_1 \dots \mathbf{s}_n$;

in the j-th stripe of samples and the target vector $\mathbf{0}^b$ or $\mathbf{u}_i$ for $i \in [b]$ and $j \in [a]$) has at least N distinct solutions with the probability at least $1 - 2/N$. Therefore, the corresponding BKW algorithm aborts with the probability at most $\frac{1}{N} \cdot c^a \cdot (\frac{1}{\gamma})^{2 \cdot c^a} \cdot \mathsf{poly}(n)$ via the union bound.

We now analyze the probability of the event that a single bit of the secret (e.g., $\mathbf{s}_1$) can be recovered correctly. Let the labels $b_1, \cdots, b_m$ (generated in line 10) have the corresponding noise $e_1, \cdots, e_m$, i.e., $b_i = \mathbf{s}_1 \oplus e_i$ for $i \in [m]$. The c-sum$^+$ subroutines are invoked $\frac{m \cdot c^a}{c-1}$ times, and each final resulting vector (that are a sum of c^a initial vectors) bears a noise of rate $\frac{1}{2} - \frac{1}{2}\gamma^{c^a}$ via the Piling-up Lemma (see Lemma 1). Moreover, $e_1, \cdots, e_m$ are all independent. Then, a single secret bit can be recovered with error rate $\frac{1}{2^n}$ (by a Chernoff Bound). Therefore, the probability of recovering secret key is $P \geq 1 - 1/N \cdot c^a \cdot (1/\gamma)^{2 \cdot c^a} \cdot \mathsf{poly}(n) - \frac{n}{2^n}$. Since it runs the c-sum$^+$ subroutine $c^a \cdot (\frac{1}{\gamma})^{2 \cdot c^a} \cdot \mathsf{poly}(n)$ times, we have $M \approx M_{c,N,b} \cdot c^a$ and $T \approx T_{c,N,b} \cdot c^a \cdot (\frac{1}{\gamma})^{2 \cdot c^a}$. □

Next, we show different variants of the c-sum$^+$ BKW via instantiating the corresponding c-sum$^+$ algorithm.

3.3 Naive c-sum$^+$ BKW Algorithm

Our naive c-sum$^+$ algorithm is showed in [33, Algorithm 3]. Similar to the naive approach [17], it first enumerates all possible $\mathbf{p} = \bigoplus_{j=1}^{c-1} \mathbf{a}_{j,i_j} \in \mathbb{F}_2^b$ for all $\mathbf{a}_{j,i_j} \in L_j$ and $j \in [c-1]$, and checks whether $\mathbf{p} \oplus \mathbf{t}$ appears in the sorted list L_c or not, where the target vector $\mathbf{t} \in \mathbb{F}_2^b$. We obtain Theorem 3 by combining Lemma 6 with Theorem 2.

Lemma 6. *The naive c-sum$^+$ algorithm solves the c-sum$^+$ problem with target length b and list size $N \geq 2^{\frac{b+1}{c-1}}$ ($2 \leq c \in \mathbb{N}$) in time $N^{c-1} \cdot \mathsf{poly}(b,c)$ and space $N \cdot \mathsf{poly}(b,c)$, and it returns N distinct solutions with the probability $1 - 2/N$.*

Proof. Sorting out the list L_c is a one-time effort that takes time $\tilde{O}(N)$, and enumerating all possible combinations of the $c-1$ lists takes time $N^{c-1} \cdot \mathsf{poly}(b,c) \cdot \log(N) = N^{c-1} \cdot \mathsf{poly}(b,c)$ where $O(b\log(N))$ accounts for the time complexity of the binary search for $\mathbf{p} \oplus \mathbf{t}$ in the sorted L_c. The algorithm consumes space of size $N \cdot \mathsf{poly}(b,c)$ since it only stores up to N solutions. □

Theorem 3 (Naive c-sum$^+$ BKW) *The $\mathsf{LPN}_{n,\mu}$ problem with $\mu = 1/2 - \gamma/2$ can be solved in time $T \approx N^{c-1} \cdot c^a \cdot (\frac{1}{\gamma})^{2 \cdot c^a}$ and space $M \approx N \cdot c^a$ with the probability $P \geq 1 - \frac{1}{N} \cdot c^a \cdot (\frac{1}{\gamma})^{2 \cdot c^a} \cdot \mathsf{poly}(n) - \frac{n}{2^n}$, where $ab \geq n$, and $N = 2^{\frac{b+1}{c-1}}$.*

Concretely, for noise rate $\mu = 1/4$, we set $a = \frac{\log(n)}{\log(c)(1+\epsilon)}$ and $b = \frac{\log(c)(1+\epsilon)n}{\log(n)}$ to get $\log(M) = \frac{\log(c)}{c-1} \cdot \frac{n(1+\epsilon)}{\log(n)}$, $\log(T) = \log(c) \cdot \frac{n(1+\epsilon+o(1))}{\log(n)}$ and $P \geq 1 - \mathsf{negl}(n)$.

3.4 Quantum c-sum$^+$ BKW Algorithm

Following the steps in [17], we adopt the Grover's algorithm [19] (see Theorem 4) to quantumly speed up the crucial (and time-consuming) first step in the naive c-sum$^+$ (see [33, Algorithm 4]). To this end, we define

$$f_{\mathbf{t}} : [N]^{c-1} \to \{0,1\}, \quad f_{\mathbf{t}} : (i_1, \cdots, i_{c-1}) \mapsto \begin{cases} 1, & \exists \mathbf{a}_{c,i_c} \in L_c : \sum_{j=1}^{c} \mathbf{a}_{j,i_j} = \mathbf{t} \\ 0 & \text{otherwise} \end{cases}.$$

Once given $(i_1, \cdots, i_{c-1}) \in f^{-1}(1)$ we can recover all i_c such that $(i_1, \cdots, i_c)$ constitutes a solution to c-sum$^+$ in time $\tilde{O}(\log(|L|))$ from a sorted list L_c. Lemma 7 follows from Theorem 4 and Lemma 4.

Theorem 4 (Grover Algorithm [9, 15, 19]) *Let $f : D \to \{0,1\}$ be a function with non-empty support. Then, Grover outputs with overwhelming probability a uniformly random preimage of 1, making q queries to f, where $q = \tilde{O}\left(\sqrt{\frac{|D|}{|f^{-1}(1)|}}\right)$.*

Lemma 7. *The quantum c-sum$^+$ algorithm solves the c-sum$^+$ problem with target length b and list size $N \geq 2^{\frac{b+1}{c-1}}$ ($2 \leq c \in \mathbb{N}$) in time $N^{\frac{c}{2}} \cdot \mathsf{poly}(b,c)$ and space $N \cdot \mathsf{poly}(b,c)$, and it returns N distinct solutions with the probability $1 - 2/N$.*

Combining Lemma 7 and Theorem 2, we obtain Theorem 5.

Theorem 5 (Quantum c-sum$^+$ BKW) *The $\mathsf{LPN}_{n,\mu}$ problem with $\mu = 1/2 - \gamma/2$ can be quantumly solved in time $T \approx N^{\frac{c}{2}} \cdot c^a \cdot (\frac{1}{\gamma})^{2 \cdot c^a}$ and space $M \approx Nc^a$ with the probability $P \geq 1 - \frac{1}{N} c^a (\frac{1}{\gamma})^{2c^a} \mathsf{poly}(n) - \frac{n}{2^n}$, where $ab \geq n$, and $N = 2^{\frac{b+1}{c-1}}$.*

Again, with noise rate $\mu = 1/4$ we set $a = \frac{\log(n)}{\log(c)(1+\epsilon)}$ and $b = \frac{\log(c)(1+\epsilon)n}{\log(n)}$ to get $\log(M) = \frac{\log(c)}{c-1} \cdot \frac{n(1+\epsilon)}{\log(n)}$, $\log(T) = \frac{c \cdot \log(c)}{2(c-1)} \cdot \frac{n(1+\epsilon+o(1))}{\log(n)}$, $P \geq 1 - \mathsf{negl}(n)$, where factor $\frac{c}{2(c-1)}$ represents the quantum speedup over the classic algorithm.

3.5 Dissection c-sum$^+$ BKW Algorithm

Esser et al. [17] borrowed the dissection technique from [14, 37] to optimize the running time of their c-sum algorithm, referred to as dissection c-sum. The dissection c-sum perfectly fits into our c-sum$^+$ problem even better with only minor adaptions. Below we briefly introduce the dissection c-sum, and analyze its running time and space consumption in solving the c-sum$^+$ problem. We defer the redundancy to the appendix and reproduced the (slightly adapted) proofs for completeness.

Following [17] we introduce the join operation (see Definition 4) to facilitate the description of the dissection c-sum algorithm. We slightly abuse the notation in Fig. 3 by extending the operation to multiple lists, e.g., $\bowtie_{\tau_3}$ operates on L_8, L_9, L_{10}, L_{11} with target τ_3. This operation can be implemented in a space friendly way without storing the intermediate lists. We simply adapt the naive $(i+1)$-sum$^+$ algorithm on lists $L_{c_{i-1}+1}, \cdots, L_{c_i}$ whose target vector τ_i may not be of full length b, in which case the algorithm returns all the combinations whose lowest $|\tau_i|$-bit sum is τ_i.

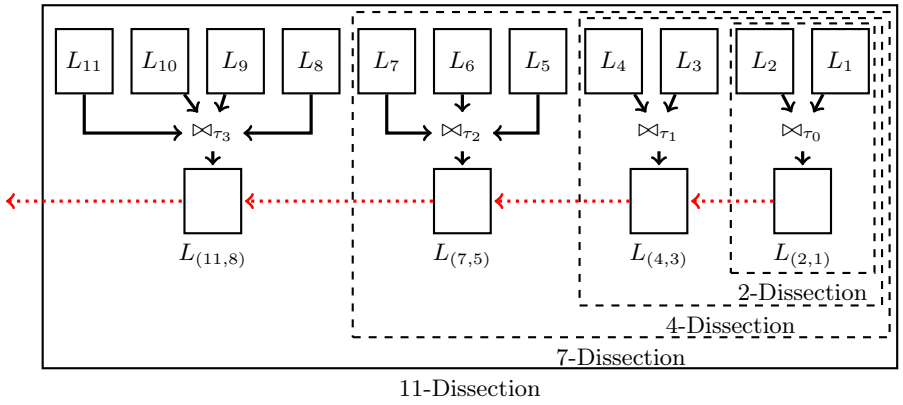


Fig. 3. An illustration of the dissection 11-sum on input lists $L_{11}, \cdots, L_1$ that recursively invokes dissection 7- and 4-sum (in dashed boxes), where $\bowtie_\tau$ is the join operator (as per Definition 4) and implemented by Naive c-sum$^+$ (as per [33, Algorithm 4]), the blank box stores the intermediate results of $\bowtie_{\tau_j}$ operation, combine results from previous invocations on-the-fly, and returns the found match through the red dotted arrows.

Definition 4 (Join Operator [17]) *Let $d \in \mathbb{N}$ and $L_1, \cdots, L_k \in (\mathbb{F}_2^d)^*$ be lists. The joins of two and multiple lists are respectively defined as*

$$L_1 \bowtie L_2 \stackrel{\mathsf{def}}{=} (\mathbf{a}_1 \oplus \mathbf{a}_2 : \mathbf{a}_1 \in L_1, \mathbf{a}_2 \in L_2) \ ,$$
$$L_1 \bowtie L_2 \bowtie \cdots \bowtie L_k \stackrel{\mathsf{def}}{=} \Big(\big((L_1 \bowtie L_2) \bowtie L_3 \big) \cdots \bowtie L_k \Big) \ .$$

For $\mathbf{t} \in \mathbb{F}_2^{d'}$ with $d' \leq d$, the join of L_1 and L_2 on target $\mathbf{t}$ is defined as

$$L_1 \bowtie_{\mathbf{t}} L_2 \stackrel{\mathsf{def}}{=} (\mathbf{a}_1 \oplus \mathbf{a}_2 : \mathbf{a}_1 \in L_1, \mathbf{a}_2 \in L_2 \wedge \mathsf{low}_{|\mathbf{t}|}(\mathbf{a}_1 \oplus \mathbf{a}_2) = \mathbf{t}) \ .$$

Definition 5 (The Magic Sequence [14]) *Let $c_{-1} \stackrel{\mathsf{def}}{=} 1$ and define the magic sequence via the recurrence $\forall i \in \mathbb{N}^+ \cup \{0\} : c_i \stackrel{\mathsf{def}}{=} c_{i-1} + i + 1$, which leads to the general formula for the magic sequence: $\mathsf{magic} \stackrel{\mathsf{def}}{=} \left\{ c_i \stackrel{\mathsf{def}}{=} \left(\frac{1}{2} \cdot (i^2 + 3i + 4) \right) \right\}_{i \in \mathbb{N}^+}$.*

The parameter c of the dissection c-sum can no longer be an arbitrary integer but belongs to the "magic sequence" (Definition 5), i.e., $c_i \stackrel{\mathsf{def}}{=} (i^2 + 3i + 4)/2$. Fix a certain i (and c_i), we recall the list size $\forall j \in [c_j] : |L_j| = N = 2^{\frac{b+1}{c_i - 1}}$. For convenience, let $\lambda \stackrel{\mathsf{def}}{=} \frac{b+1}{c_i - 1}$ so that block size $b = (c_i - 1)\lambda - 1$. The algorithm employs the meet-in-the-middle strategy with (intermediate) targets of smaller sizes $\tau_j \in \mathbb{F}_2^{j\lambda}$ (for $j \in [i]$), and $\tau_0 \in \mathbb{F}_2^{\lambda}$ in its iterations.

We now give a high-level recursive description about the Dissection c_i-sum algorithm that aims to find out N solutions to the c_i-sum$^+$ problem for a target $\mathbf{t} \in \mathbb{F}_2^b$, which recursively invokes the dissection c_j-sum algorithm ($j < i$) to get all the combinations whose lowest $j\lambda$-bit sum is τ_j. The base case ($i = 0$, $c_0 = 2$), i.e., the Dissection 2-sum degenerates into the naive 2-sum$^+$ algorithm with a minor exception that the target τ_0 may be not of full length b. We illustrate the general case with a concrete example ($i = 3$, $c_3 = 11$) in Fig. 3. Taking as input lists $L_1, \cdots, L_{c_i}$ and a target $\mathbf{t}$, the algorithm divides the lists into two groups $L_1, \cdots, L_{c_{i-1}}$ and $L_{c_{i-1}+1}, \cdots, L_{c_i}$, where $c_i = c_{i-1} + i + 1$ due to the magic sequence. For each intermediate target $\tau_i \in \mathbb{F}_2^{i \cdot \lambda}$, do the following:

1. Invoke the (adapted) naive $(i+1)$-sum$^+$ algorithm on lists $L_{c_{i-1}+1}, \cdots, L_{c_i}$ with the target vector τ_i to get all the combinations whose lowest $(i \cdot \lambda)$-bit sum is τ_i. Store all the solutions in list $L_{(c_i, c_{i-1}+1)}$.
2. Invoke the dissection $c_{(i-1)}$-sum algorithm on lists $L_1, \cdots, L_{c_{i-1}}$ with target $\mathsf{low}_{(i-1)\cdot\lambda}(\tau_i) \oplus \mathsf{low}_{(i-1)\cdot\lambda}(\mathbf{t})$. The results are passed to the parent call on-the-fly (see the red dotted line in Fig. 3), and combined with those in $L_{(c_i, c_{i-1}+1)}$, producing only those summing to $\mathbf{t}$ as output.
3. Repeat the above for all possible values of $\tau_i \in \mathbb{F}_2^{i \cdot \lambda}$.

On space consumption. We stress that the above provides only an oversimplified description, and the actual algorithm (see [33, Algorithm 6 & 7]) is slightly

more complicated to keep the space consumption within $O(iN)$. First, for each $0 \le j \le i$ we use $L_{(c_j, c_{j-1}+1)}$ to store the results of the naive $(j+1)$-sum$^+$ on lists $L_{c_{j-1}+1}$, ..., L_{c_j} (see the $\bowtie_{\tau_j}$ operation and the blank boxes in Fig. 3). Second, every single result from $L_{(2,1)}$ is passed to $L_{(4,3)}$, and so on, all the way to $L_{(c_i, c_{i-1}+1)}$ on-the-fly to form the final output (or be discarded if it fails the checking). In other words, no additional space will be allocated for merging $L_{(2,1)}$ with $L_{(4,3)}$, and then $L_{(7,5)}$, etc., to avoid a blowup in space consumption. Finally, one can observe that the intermediate target size τ_j $(0 \le j \le i)$ are chosen such that the expected size of $L_{(c_j, c_{j-1}+1)}$ is N. That is, $(j+1)$-sum$^+$ on $(j+1)$ lists, each of size $N = 2^\lambda$, yields N^{j+1} combinations, each having a chance of $2^{-|\tau_j|}$ to hit target τ_j. Thus, we have $N^{j+1}/2^{|\tau_j|} = N$ (more formally in the full version [33]), and the overall space consumption is $O(iN)$.

The dissection c_i-sum$^+$ [33, Algorithm 6] invokes the interative procedure c_j-Dissect [33, Algorithm 7] for $j \le i$ to solve the c_i-sum$^+$ problem for $c_i \in \mathsf{magic}$. We already show in Lemma 4 that for any $2 \le c \in \mathbb{N}$ the problem has at least N solutions (except with the probability $2/N$). Esser et al. [17] showed that the dissection c_i-sum$^+$ does an exhaustive search over all solutions.

Compared with the naive c-sum$^+$ algorithm that also exhausts all solutions, dissection c_i-sum$^+$ enjoys optimized time complexity as stated in [33, Lemma 17]. Esser et al. [17] analyzed the c_i-Dissect [33, Algorithm 7] subroutine (essentially the $\bowtie_{\tau_j}$ operation in Fig. 3) in terms of expected time and space, and we further give their upper bounds in [33, Lemma 14 & 16] to reach a more formal statement in [33, Lemma 17]. Combining [33, Lemma 17] and Theorem 2, we obtain Theorem 6.

Theorem 6 (Dissection c-sum$^+$ BKW Algorithm) *For any $c_i \in \mathsf{magic}$, the $\mathsf{LPN}_{n,\mu}$ problem with $\mu = 1/2 - \gamma/2$ can be solved in time $T \approx N^{c_{i-1}} \cdot c_i^a \cdot (\frac{1}{\gamma})^{2 \cdot c_i^a}$ and space $M \approx N \cdot c_i^a$ with the probability $P \ge 1 - \frac{1}{N} \cdot c_i^a \cdot (\frac{1}{\gamma})^{2 \cdot c_i^a} \cdot \mathsf{poly}(n) - \frac{n}{2^n}$, where $ab \ge n$, and $N = 2^{\frac{b+1}{c_i - 1}}$.*

Concretely, for $\mu = 1/4$, we can set $a = \frac{\log(n)}{\log(c_i)(1+\epsilon)}$ and $b = \frac{\log(c_i)(1+\epsilon)n}{\log(n)}$ so that $\log(M) = \frac{\log(c_i)}{c_i - 1} \cdot \frac{n(1+\epsilon)}{\log(n)}$, $\log(T) = (1 - \frac{i}{c_i - 1}) \cdot \log(c_i) \cdot \frac{n(1+\epsilon+o(1))}{\log(n)}$, $P \ge 1 - \mathsf{negl}(n)$, where the optimization over the naive c-sum$^+$ BKW is highlighted.

3.6 Tailored Dissection c-sum$^+$ BKW

The dissection c-sum$^+$ trades time for space of smaller size $M_i \approx 2^{\left(\frac{\log(c_i)}{c_i - 1}\right) \frac{n(1+\epsilon)}{\log(n)}}$ where $c_i = (i^2 + 3i + 4)/2$. In practice, it may turn out that the size of actual usable space $M \in (M_i, M_{i-1})$, leaving an unused space of size $(M_{i-1} - M)$. To address this issue, Esser et al. [17] introduced the tailored dissection c_i-sum technique to enable more fine-grained time-space tradeoffs. That is, still use $N = 2^{\frac{b+1}{c_i - 1}}$, but increase the list size 2^λ from N to $N^\beta \approx M$ $(\beta > 1)$ to fully utilize the available space. However, the optimized running time of their algorithm needs not only the independence heuristic but also relies on the

tailoring heuristic [17] (see [33, Appendix B], which postulates that one needs only to go through the first 2^y (for $y = b - c_{i-1} \cdot \lambda + 1$) constraints $\tau_i \in \mathbb{F}_2^{i \cdot \lambda}$ (in the outmost for-loop of [33, Algorithm 7]) to recover at least N^β distinct solutions (with high probability). In a similar vein, we present an unconditional version called tailored dissection c_i-sum$^+$ that aims for the first N^β (instead of all) distinct solutions and halts as soon as $2^\lambda = N^\beta$ solutions are found (see line 9 of [33, Algorithm 7]). Instead of relying on any heuristics, we prove in [33, Lemma 18] unconditionally that the outmost for-loop needs only 2^y iterations for $y = b - c_{i-1}\lambda + 1$. Combining [33, Lemma 19] and Theorem 2, we obtain Theorem 7.

Theorem 7 (Tailored Dissection c-sum$^+$ BKW) *For any $c_i \in \mathsf{magic}$, the $\mathsf{LPN}_{n,\mu}$ problem with $\mu = 1/2 - \gamma/2$ can be solved in time $T \approx N^{c_{i-1}+(1-\beta)\cdot i} \cdot c_i^a \cdot (\frac{1}{\gamma})^{2 \cdot c_i^a}$ and space $M \approx N^\beta \cdot c_i^a$ with the probability $P \geq 1 - \frac{1}{N^\beta} \cdot c_i^a \cdot (\frac{1}{\gamma})^{2 \cdot c_i^a} \cdot \mathsf{poly}(n) - \frac{n}{2^n}$, where $ab \geq n$, $N = 2^{\frac{b+1}{c_i - 1}}$ and $\beta \in [1, \frac{c_i - 1}{c_{i-1}}]$.*

Concretely, for $\mu = 1/4$ we can set $a = \frac{\log(n)}{\log(c_i)(1+\epsilon)}$ and $b = \frac{\log(c_i)(1+\epsilon)n}{\log(n)}$ so that $\log(M) = \frac{\beta \cdot \log(c_i)}{c_i - 1} \cdot \frac{n(1+\epsilon)}{\log(n)}$, $\log(T) = (1 - \frac{\beta \cdot i}{c_i - 1}) \cdot \log(c_i) \cdot \frac{n(1+\epsilon+o(1))}{\log(n)}$ and $P = 1 - \mathsf{negl}(n)$ where the difference to the dissection c-sum$^+$ BKW was highlighted.

3.7 Time-space Trade-offs for solving LWE

Regev [36] introduced the Learning With Errors (LWE) problem, generalizing LPN over arbitrarily large moduli in presence of Gaussian-like noise.

Definition 6 (Learning With Errors) *Let $\mathcal{D}_\sigma$ be a discrete Gaussian distribution with mean zero and variance σ^2. For $n \in \mathbb{N}$, prime $p \in \mathbb{N}$, $\mathbf{s} \in \mathbb{F}_p^n$, denote by Sample an oracle that, when queried, samples $\mathbf{a} \xleftarrow{\$} \mathbb{F}_p^n$, $e \leftarrow \mathcal{D}_\sigma$ and outputs a sample of the form $(\mathbf{a}, l) := (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$. The $\mathsf{LWE}_{n,\sigma,p}$ problem refers to recovering the random secret $\mathbf{s}$ given access to Sample.*

Albrecht et al. [1] adapted the BKW algorithm to solve the LWE problem, with subsequent improvement by [10, 21, 24]. Similarly, the BKW reduces the dimension of LWE by summing up samples and cancelling out the corresponding blocks in iterations. The number of samples needed for the majority vote is $m = \mathrm{e}^{\frac{4\pi^2\sigma^2 2^a}{p^2}}$ after a BKW steps [28]. Herold et al. [24] showed that setting $a = (1 - \epsilon_a)\log(n) + 2\log(p) - 2\log(\sigma)$ for constant $\epsilon_a > 0$ yields $m = \mathrm{e}^{4\pi^2 n^{1-\epsilon_a}}$ and results in time, space and sample complexities $\tilde{O}\big(p^b \cdot \mathrm{e}^{4\pi^2 n^{1-\epsilon_a}}\big) = p^{b \cdot (1+\epsilon)} = 2^{\frac{n \cdot \log(p) \cdot (1+\epsilon)}{\log(n) + 2\log(p) - 2\log(\sigma)}}$.

Following the steps of Esser et al. [17], we also generalize the c-sum$^+$ problem to arbitrary moduli p and employed (slightly tweaked versions of) the aforementioned algorithms to solve the c-sum$^+$ problem with arbitrary moduli p whose elementary operations (e.g., addition, sorting and binary search) are

now over $\mathbb{F}_p$. Compared with [24], we adjust a by a factor of $\log(c)$ and set $a = \frac{(1-\epsilon_a)\log(n)+2\log(p)-2\log(\sigma)}{\log(c)}$ for constant $\epsilon_a > 0$. We summarize the results in Table 5, which are essentially the same as that of the c-sum BKW for LWE [17] but without using heuristics.

Table 5. The time and space complexities of the c-sum (c-sum$^+$) BKW algorithms for solving the $\mathsf{LWE}_{n,\sigma,p}$ problem, where $N_c = 2^{\frac{\log(c)}{c-1}\cdot\frac{n\cdot\log(p)\cdot(1+\epsilon)}{\log(n)+2\log(p)-2\log(\sigma)}}$, n is the dimension, and constant $\epsilon > 0$.

	c-sum (c-sum$^+$) BKW	Space	Time	for
Classic	Original BKW [6]	N_2	N_2	$c = 2$
	Naive	N_c	N_c^{c-1}	$c \geq 2$
	Dissection	N_c	$N_c^{c-\sqrt{2c}}$	$c \in \mathsf{magic}$
	Tailored Dissection	N_c^{β}	$N_c^{c-\beta\sqrt{2c}}$	$c \in \mathsf{magic}$, $\beta \in [1, \frac{\sqrt{c}}{\sqrt{c}-1}]$
Quantum	Naive + Grover	N_c	$N_c^{c/2}$	$c \geq 2$

4 The c-sum# BKW with Time/Sample Optimizations

In this section, we consider the extended k-Generalized Birthday Problem (see Section 3), and give the full-fledged variant, called c-sum# BKW, to optimize the time, space and sample complexities of the original BKW algorithm [6]. Moreover, it further pushes the sample complexity to 2^{n^ϵ} or even $n^{1+\epsilon}$, which also optimize the complexities over Lyubashevsky's BKW variant [34].

4.1 Time, Space, and Sample Optimizations

As shown in Table 6, we compare the results of the original BKW [6], Devadas et al.'s optimized version [12] and our c-sum# BKW (for $c = 2$ as in Theorem 8).

Table 6. The space, time and sample complexities of different variants of the BKW algorithms for solving the $\mathsf{LPN}_{n,\mu}$ problem with $\mu = (1-\gamma)/2$, $\gamma \geq 2^{-n^\sigma}$ and constant $0 < \sigma < 1$ under condition $N_1 \approx N_2$, where $ab = n$, $N_1 = 2^b$ and $N_2 = (1/\gamma)^{2^{a+1}}$ disregarding $\mathsf{poly}(n)$ factors for convenience.

Algorithm	Space	Time	Sample	Condition
The original BKW [6]	N_1	$N_1 \cdot N_2$	$N_1 \cdot N_2$	$N_1 \approx N_2$
Devadas et al.'s [12]	$N_1 \cdot \sqrt{N_2}$	$N_1 \cdot \sqrt{N_2}$	N_1	$N_1 \approx N_2$
Our 2-sum# BKW	N_1	N_1	N_1	$N_1 \approx N_2$

We know that the last step of the BKW involves balancing the two factors $N_1 = 2^b$ and $N_2 = (1/\gamma)^{2^{a+1}}$ to roughly the same magnitude given $ab = n$. As

specified in 2-sum$^{\#}$ BKW (see Theorem 8 for $c = 2$), it requires essentially the same condition, i.e., $b = 2^{a+1}\log(1/\gamma) + O(\log(n))$. Asymptotically, for constant $0 < \gamma < 1$, we typically set $a = \frac{\log(n)}{1+\epsilon}$ and $b = \frac{(1+\epsilon)n}{\log(n)}$, and thus our algorithm speeds up the running time of the original BKW [6] by a factor of $2^{n\frac{1}{1+\epsilon}}$ while using roughly the same amount of space, where constant ϵ is arbitrarily close to 0 for optimized time complexity. Recently, Devadas et al. [12] optimized the running time of the orginal BKW from $N_1 \cdot N_2$ to $N_1 \cdot \sqrt{N_2}$ at the cost of increasing the space complexity from N_1 to $N_1 \cdot \sqrt{N_2}$. Thus, the 2-sum$^{\#}$ BKW enjoys a sub-exponential factor advantage both in time/space complexities compared to [12].

Algorithm 3: The c-sum$^{\#}$ BKW

Input: access to the oracle $\mathsf{LPN}_{n,\mu}$
Output: $\mathbf{s} \in \mathbb{F}_2^n$

1 $b := \frac{n}{a}$, $N := 2^{\frac{b}{c-1}}$;
2 Save fresh LPN samples in $L_{0,1}$, ..., L_{0,c^a}, each of size N;
3 **for** $j \leftarrow 1, \cdots, a-1$ **do**
4 **for** $k \leftarrow 1, \cdots, c^{a-j}$ **do**
5 $L_{j,k} \leftarrow c\text{-sum}^{+}(L_{j-1,c(k-1)+1}, \cdots, L_{j-1,ck}, j, 0^b)$;
6 $L_{a,1} \leftarrow c\text{-sum}^{+}(L_{a-1,1}, \cdots, L_{a-1,c}, a, \mathbf{u}_1)$;
7 $\mathbf{s}_1 \leftarrow \text{majorityvote}(b_1, \cdots b_{|L_{a,1}|})$;
8 Determine $\mathbf{s}_2, \cdots, \mathbf{s}_n$ the same way over the same LPN samples;
9 **Return** $\mathbf{s} = \mathbf{s}_1 \ldots \mathbf{s}_n$;

Majority voting on correlated samples. The c-sum BKW [17] and our c-sum^{+} BKW (Algorithm 2) pick a single sample from $L_{a,1}$ and repeat the process for $m \approx (1/\gamma)^{2^{a+1}}$ times on fresh LPN samples (see line 2-10 in Algorithm 2). We argue that this step can be removed with a careful adaption, and therefore reduces the time/sample complexities by factor $2^{\Omega(n\frac{1}{1+\epsilon})}$. This is the motivation of introducing the extended k-generalized birthday problem (see Section 3). Hopefully, we recover the single bit of secret via a majority voting on the elements in $L_{a,1}$ (line 7 in Algorithm 3). This is non-trivial since the noise bits in $L_{a,1}$ are linear combinations of individual noises of the LPN samples, and thus they are not even pairwise independent [5]. We observe that in order to majority-vote for the correct result it suffices that the resulting noise remains biased-to-zero. For every sample list $L_{j,k}$ we define the corresponding noise-indicator list $E_{j,k}$, whose every i-th element $(-1)^{e_i}$ corresponds to the i-th element of $L_{j,k}$, i.e., $(\mathbf{a}_i, \mathbf{a}_i \cdot \mathbf{s} \oplus e_i)$. $\mathsf{bias}(E_{j,k}) = \sum_{i=1}^{|E_{j,k}|}(-1)^{e_i}$ refers to the

[5] Unlike uniformly random vectors, the linear combinations of i.i.d. biased bits are not pairwise independent, e.g., $e_1 + e_2$ and e_2 for $e_1, e_2 \leftarrow \mathcal{B}_\mu$ with $0 < \mu < 0.5$.

difference between the numbers of 0's and 1's in the noise of $L_{j,k}$. Therefore, the majority voting is successful if and only if the final $\mathsf{bias}(E_{a,1}) > 0$.

THE c-SUM$^{\#}$ BKW. We now describe how to adapt the c-sum$^+$ BKW (Algorithm 3) to avoid the outmost repeat-m-times loop. The c-sum$^+$ BKW is sample-preserving, i.e., it invokes subroutines such as the naive c-sum$^+$ [33, Algorithm 4] that halt as soon as N solutions are found. In contrast, we let the c-sum$^{\#}$ BKW be exhaustive, i.e., the underlying c-sum$^+$ solver (e.g., [33, Algorithm 9]) must output all solutions. We start with the initial leaf-level lists $E_{0,1}, \cdots, E_{0,c^a}$ with $|E_{0,k}| = N$ and sufficiently large $\mathsf{bias}(E_{0,k})$ for every $k \in [c^a]$. Then, as shown in Lemma 9, for every $j \in [a]$ and $k \in [c^{a-j}]$ the $|E_{j,k}|$ will be bounded within $N(1 \pm o(1))$ and $\mathsf{bias}(E_{j,k})$ stays positive. To achieve this, we set $N = 2^{b/(c-1)}$ (instead of $N = 2^{(b+1)/(c-1)}$). Consider the c-sum$^+$ problem instance whose input noise-indicator lists are $E_{j-1,1}, \cdots, E_{j-1,c}$ and output noise-indicator list $E_{j,1}$, whose elements are chosen from $JE_{j,1} \stackrel{\mathsf{def}}{=} E_{j-1,1} \bowtie \cdots \bowtie E_{j-1,c}$ (all possible c-sums). In particular, each element from list $JE_{j,1}$ is included into $E_{j,1}$ iff the corresponding c-sum$^+$ hits the target, which occurs with the probability 2^{-b}. Further, whether an element in $JE_{j,1}$ hits the specified target or not is a pairwise independent event (see Lemma 5). With $|E_{j-1,k}| \approx N$ for every $k \in [c]$, we have that $|E_{j,1}|$ has expected value roughly $N^c/2^b = N$ and thus remains around N by Chebyshev's inequality. We also lower bound the corresponding $\mathsf{bias}(E_{j,1})$ for every $j \in [a]$. We state the results in Lemma 9, and prior to that we introduce Lemma 8 as an analogue of the piling-up lemma that characterizes how the bias is amplified through the c-sum$^+$ operations.

Lemma 8. *For $JE_{j+1} \stackrel{\mathsf{def}}{=} E_{j,k+1} \bowtie E_{j,k+2} \cdots \bowtie E_{j,k+c}$, we have $\mathsf{bias}(JE_{j+1}) = \prod_{i=1}^{c} \mathsf{bias}(E_{j,k+i})$.*

Proof. It follows from the definitions of bias and $\bowtie$ by rearranging the terms:

$$\begin{aligned}\mathsf{bias}(JE_{j+1}) &= \sum_{l_1 \in [n_1], \cdots, l_c \in [n_c]} (-1)^{e^1_{l_1}} \times \cdots \times (-1)^{e^c_{l_c}} \\ &= \Big(\sum_{l_1 \in [n_1]} (-1)^{e^1_{l_1}} \Big) \times \cdots \times \Big(\sum_{l_c \in [n_c]} (-1)^{e^c_{l_c}} \Big) = \prod_{i=1}^{c} \mathsf{bias}(E_{j,k+i}) \ ,\end{aligned}$$

where we use shorthand $n_i \stackrel{\mathsf{def}}{=} |E_{j,k+i}|$ for $1 \le i \le c$ for notational convenience. □

Lemma 9. *For $N = 2^{\frac{b}{c-1}}$, any $2 \le c \in \mathbb{N}$, $0 < \varepsilon < 1$ and $0 < \delta < 1$ such that $\delta^{c^a}\sqrt{N}\varepsilon \ge 2^a c^{2a}$, if the level-0 lists $E_{0,1}$, …, E_{0,c^a} satisfy $|E_{0,k}| = N$, $\mathsf{bias}(E_{0,k}) \ge \delta N$ for $1 \le k \le c^a$. Then, at every level $j \in [a]$, for every k-th list $E_{j,k}$ $(1 \le k \le c^{a-j})$ we have*

$$\Pr\Big[\mathsf{bias}(E_{j,k}) \le \big(\delta^{c^j} N - \frac{2^j\sqrt{N}c^{2j}}{\varepsilon}\big)\Big] \le c^{4j} \cdot \varepsilon \ ,$$

$$\Pr\Big[\big||E_{j,k}| - N\big| \ge \frac{2^j\sqrt{N}c^{2j}}{\varepsilon}\Big] \le c^{4j} \cdot \varepsilon \ .$$

Proof. The base case $j = 0$ holds by assumption, i.e., $\mathsf{bias}(E_{0,k}) \geq \delta N$ and $|E_{0,k}| = N$ for every $1 \leq k \leq c^a$. We prove the rest by induction, i.e., if it holds for level j, then it also true for level $j+1$. It suffices to consider the first list $E_{j+1,1}$ on level $j+1$ whose elements are selected from the set of all c-sum$^+$ of the c lists, i.e., $JE_{j+1,1}{=}E_{j,1} \bowtie \cdots \bowtie E_{j,c}$. with the probability at least $1 - c^{4j+1}\varepsilon$, we have (by the definition of $\bowtie$) $N^c(1 - \frac{2^j c^{2j+1}}{\sqrt{N}\varepsilon}) \leq N^c(1 - \frac{2^j c^{2j}}{\sqrt{N}\varepsilon})^c < |JE_{j+1,1}| < N^c(1 + \frac{2^j c^{2j}}{\sqrt{N}\varepsilon})^c \leq N^c(1 + \frac{2^{j+1} c^{2j+1}}{\sqrt{N}\varepsilon})$, where by [33, Lemma 21] $(1+d)^c \leq 1 + 2cd$ and $(1-d)^c \geq 1 - cd$ for $0 < cd < 1$, $c \geq 2$. Every element from list $JE_{j+1,1}$ has a chance of 2^{-b} to be selected into $E_{j+1,1}$ in a pair-wise independent manner among the elements of $JE_{j+1,1}$ (see Lemma 5). Thus, the above implies (recall $N^{c-1} = 2^b$) $\Pr\Big[\big|\mathbb{E}\big[|E_{j+1,1}|\big] - N\big| < \frac{2^{j+1}\sqrt{N}c^{2j+1}}{\varepsilon}\Big] \geq 1 - c^{4j+1}\varepsilon$. Similar to the proof of Lemma 4 (except for a different value of N), we have

$$\begin{aligned}
&\Pr\Big[\big||E_{j+1,1}| - N\big| \geq \frac{2^{j+1}\sqrt{N}c^{2j+2}}{\varepsilon}\Big] \\
\leq &\Pr\Big[\Big||E_{j+1,1}| - \mathbb{E}\big[|E_{j+1,1}|\big]\Big| \geq \frac{2^{j+1}\sqrt{N}c^{2j+1}(c-1)}{\varepsilon}\Big] \\
&+ \Pr\Big[\Big|\mathbb{E}\big[|E_{j+1,1}|\big] - N\Big| \geq \frac{2^{j+1}\sqrt{N}c^{2j+1}}{\varepsilon}\Big] \\
\leq &\frac{Var\big[|E_{j+1,1}|\big]}{N/\varepsilon^2} + c^{4j+1}\cdot\varepsilon \leq \frac{\mathbb{E}\big[|E_{j+1,k}|\big]}{N/\varepsilon^2} + c^{4j+1}\cdot\varepsilon \leq c^{4j+3}\cdot\varepsilon \ .
\end{aligned} \tag{1}$$

By Lemma 8 the following holds with the probability at least $1 - c^{4j+1}\varepsilon$

$$\mathsf{bias}(JE_{j+1,1}) > \delta^{c^{j+1}} N^c\big(1 - \frac{2^j c^{2j}}{\delta^{c^j}\sqrt{N}\varepsilon}\big)^c \geq \delta^{c^{j+1}} N^c\big(1 - \frac{2^j c^{2j+1}}{\delta^{c^j}\sqrt{N}\varepsilon}\big) \ ,$$

where the Bernoulli's inequality $(1-d)^c \geq 1 - cd$ is applicable since $c \geq 2$ and $d = \frac{2^j c^{2j}}{\delta^{c^j}\sqrt{N}\varepsilon} < \frac{2^a c^{2a}}{\delta^{c^a}\sqrt{N}\varepsilon} \leq 1$. We recall $\mathsf{bias}(E_{j+1,1}) \stackrel{\mathsf{def}}{=} \sum_{l=1}^{|JE_{j+1,1}|} \mathbf{v}_l \cdot (-1)^{e_l}$, where random variable $\mathbf{v}_l$ is 1 if the corresponding c-sum$^+$ hits the specified target (so that the corresponding $(-1)^{e_l}$ is included in $E_{j+1,1}$) or is 0 otherwise. By Lemma 5 all the $\mathbf{v}_l$'s are pairwise independent, each with expectation 2^{-b}, and therefore $\mathbb{E}[\mathsf{bias}(E_{j+1,1})] = 2^{-b} \cdot \mathsf{bias}(JE_{j+1,1})$. We have $\Pr\Big[\mathbb{E}\big[\mathsf{bias}(E_{j+1,1})\big] > \delta^{c^{j+1}} N - \frac{2^j\sqrt{N}c^{2j+1}}{\varepsilon}\Big] \geq 1 - c^{4j+1}\varepsilon$, and thus

$$\begin{aligned}
&\Pr\Big[\mathsf{bias}(E_{j+1,1}) \leq \delta^{c^{j+1}} N - \frac{2^{j+1}\sqrt{N}c^{2j+2}}{\varepsilon}\Big] \\
\leq &\Pr\Big[\mathsf{bias}(E_{j+1,1}) - \mathbb{E}\big[\mathsf{bias}(E_{j+1,1})\big] \leq \frac{2^j\sqrt{N}c^{2j+1}(2c-1)}{\varepsilon}\Big] \\
&+ \Pr\Big[\mathbb{E}\big[\mathsf{bias}(E_{j+1,1})\big] < \delta^{c^{j+1}} N - \frac{2^j\sqrt{N}c^{2j+1}}{\varepsilon}\Big] \\
\leq &\frac{Var\big[\mathsf{bias}(E_{j+1,1})\big]}{N/\varepsilon^2} + c^{4j+1}\cdot\varepsilon \leq \frac{\mathbb{E}\big[|E_{j+1,k}|\big]}{N/\varepsilon^2} + c^{4j+1}\cdot\varepsilon \leq c^{4j+3}\cdot\varepsilon \ ,
\end{aligned} \tag{2}$$

where the analysis is essential the same as that for bounding $|E_{j+1,1}|$ except that $Var\big[\mathsf{bias}(E_{j+1,1})\big] = \sum_{l=1}^{|JE_{j+1,1}|} Var\big[\mathbf{v}_l \cdot (-1)^{e_l}\big] \le \sum_{l=1}^{|JE_{j+1,1}|} \mathbb{E}\big[\mathbf{v}_l\big] = \mathbb{E}\Big[\sum_{l=1}^{|JE_{j+1,1}|} \mathbf{v}_l\Big]$. □

Now we state the fully optimized algorithm in Theorem 8, and compare the case $c = 2$ (no time-space tradeoff) with the original BKW [6] and the one by Devadas et al. [12] in Table 6.

Theorem 8 (The c-sum$^{\#}$ BKW) *The* $\mathsf{LPN}_{n,\mu}$ *problem with* $\mu = 1/2 - \gamma/2$ *can be solved in time* T *and space* M *with the probability* P *as below*

$$T \approx T_{c,N,b} \cdot c^a,\ M \approx M_{c,N,b} \cdot c^a,\ P \ge 1 - 2c^{5a} \cdot n \cdot \varepsilon,$$

where $T_{c,N,b}$ *and* $M_{c,N,b}$ *are respectively the time and space complexities of the* c-sum^{+} *algorithm that aims for all distinct solutions to the* c-sum^{+} *problem with block (target) size* b, $ab \ge n$, $b > n^{0.6}$, $\gamma > 2^{-b/3}$, $b/\big(2(c-1)\big) \ge c^a \log(1/\gamma) + 3a\log(c) + 2\log(1/\varepsilon) + \mathsf{negl}(n)$ *and* $N = 2^{\frac{b}{c-1}}$ *for* $2 \le c \in \mathbb{N}$.

Notice: for now we omit the sample complexity since $Q \approx M$ *under the scenario of unlimited samples.*

Proof. Set the δ in Lemma 9 to $\gamma - 2^{-\frac{b}{2}}\sqrt{\log(1/\varepsilon)}$, and we have by Chernoff bound

$$\Pr\Big[\mathsf{bias}(E^0_{0,k}) \le N \cdot \delta\Big] \le \Pr\Big[\mathsf{bias}(E^0_{0,k})/N - \gamma \le (\delta - \gamma)\Big] \le 2^{-2^{-b}\log(1/\varepsilon)N} = \varepsilon\ ,$$

where $N = 2^{\frac{b}{c-1}}$. The condition $\delta^{c^a}\sqrt{N}\varepsilon \ge 2^a c^{2a}$ in Lemma 9 is now

$$\begin{aligned}
\frac{b}{2(c-1)} &\ge c^a \log(1/\delta) + a + 2a\log(c) + \log(1/\varepsilon) \\
&= c^a \log(1/\gamma) + a + 2a\log(c) + \log(1/\varepsilon) + c^a \log\Big(1 + \frac{2^{-b/2} O(\sqrt{\log(1/\varepsilon)})}{\gamma}\Big) \\
&\ge c^a \log(1/\gamma) + a + 2a\log(c) + \log(1/\varepsilon) + c^a 2^{-b/6} \cdot O\big(\sqrt{(1/\varepsilon)}\big)\ .
\end{aligned}$$

By Lemma 9 the size of every list $E_{j,k}$ is at most $N + N^{0.5} \cdot c^{3a}/\varepsilon = O(N)$ with the probability at least $1 - c^{4a} \cdot \varepsilon$, and thus all lists have size $O(N)$ with the probability at least $1 - c^{5a} \cdot n \cdot \varepsilon$. As for the correctness, the bias of the final list $E_{a,1}$ is positive with the probability at least $1 - c^{4a} \cdot \varepsilon$ in order to successfully recover a single bit of the secret. Overall, it recovers the whole secret correctly with the probability more than $1 - c^{4a} \cdot n \cdot \varepsilon$ by the union bound. □

4.2 Sample Reduction for BKW

Lyubashevsky [34] introduced the "sample amplification" technique to further push the sample complexity to $Q = n^{1+\epsilon}$. Let $(\mathbf{A}, \mathbf{t}^{\mathsf{T}} = (\mathbf{s}^{\mathsf{T}}\mathbf{A} + \mathbf{x}^{\mathsf{T}}))$ be all the

LPN samples one can have, where $\mathbf{A}$ is the $n \times Q$ matrix, and vectors with 'T' denote row vectors. A "sample amplification" oracle takes as input $(\mathbf{A}, \mathbf{t}^\mathsf{T})$ and responds with $(\mathbf{A}\mathbf{r}_i, \mathbf{t}^\mathsf{T}\mathbf{r}_i = \mathbf{s}^\mathsf{T}\mathbf{A}\mathbf{r}_i + \mathbf{x}^\mathsf{T}\mathbf{r}_i)$ as the i-th re-randomized LPN sample, and generates as many LPN sample as needed, where every $\mathbf{r}_i \xleftarrow{\$} \mathcal{R}_{Q,w}$ is drawn from the set of length-Q-weight-w strings uniformly at random. Finally, invoke the original BKW [6] on the generated samples. In order to make the approach work provably, $(\mathbf{A}, \mathbf{A}\mathbf{r}_i, \mathbf{x}^\mathsf{T}\mathbf{r}_i)$ should be statistically close to $(\mathbf{A}, \mathbf{U}_n, \mathbf{x}^\mathsf{T}\mathbf{r}_i)$ by the leftover hash lemma [25], which requires min-entropy $\mathbf{H}_\infty(\mathbf{r}_i) = \log \binom{Q}{w} > n$. Therefore, Lyubashevsky [34] chose $w = \frac{2n}{\epsilon \log(n)}$ for $Q = n^{1+\epsilon}$.

Our c-sum$^\#$ BKW supports sample amplification in a different and slightly more efficient way. The c-sum$^\#$ BKW (Algorithm 3) initializes the lists $L_{0,1}, \ldots, L_{0,c^a}$, with independent fresh LPN samples. However, the pairwise independence preserving lemma (Lemma 5) only requires each $L_{0,k}$ (for $k \in [2^a]$) has pairwise independent vectors. Our sample amplification simply divides $\mathbf{A}$ into $n \times \frac{Q}{2^a}$ sub-matrices $\mathbf{A}_1, \cdots, \mathbf{A}_{2^a}$ accordingly, and loads each $L_{0,k}$ with distinct w-linear combinations of the $(\mathbf{A}_k, \mathbf{s}^\mathsf{T}\mathbf{A}_k + \mathbf{x}_k^\mathsf{T})$, i.e.,

$$\forall k \in [2^a] : L_{0,k} := \Big((\mathbf{A}_k\mathbf{r}_1, \mathbf{s}^\mathsf{T}\mathbf{A}_k\mathbf{r}_1 + \mathbf{x}_k^\mathsf{T}\mathbf{r}_1), \cdots, (\mathbf{A}_k\mathbf{r}_N, \mathbf{s}^\mathsf{T}\mathbf{A}_k\mathbf{r}_N + \mathbf{x}_k^\mathsf{T}\mathbf{r}_N) \Big)$$

where $\mathbf{r}_1, \cdots, \mathbf{r}_N$ are distinct vectors of weight w, and $N = 2^b \le \binom{Q/2^a}{w}$. So far we essentially override the LPN sample oracle of the c-sum$^\#$ BKW (line 2 of Algorithm 3), which takes time and space 2^{a+b}. The rest of the steps are the same as those in Algorithm 3.

Lemma 10. *For $k = o(m)$ we have $\log \binom{m}{k} = (1 + o(1))k \log \left(\frac{m}{k}\right)$.*

Lemma 11 ([34]). *If a bucket contains m balls, $(\frac{1}{2} + p)m$ of which are colored white, and the rest colored black, and we select k balls at random without replacement, then the probability that we selected an even number of black balls is at least $\frac{1}{2} + \frac{1}{2}\left(\frac{2mp-k+1}{m-k+1}\right)^k$.*

Theorem 9 (The 2-sum$^\#$ BKW with fewer samples) *The $\mathsf{LPN}_{n,\mu}$ problem with $\mu = 1/2 - \gamma/2$ and given up to Q samples can be solved in time T, space M with the probability P as below*

$$T \approx 2^{a+b} \ , \quad M \approx 2^{a+b} \ , \quad P \ge 1 - 2^{6a} \cdot n \cdot \varepsilon - 2^a \cdot 2^{-\Omega(\frac{Q\gamma^2}{2^a})} \ ,$$

where $a, b, w \in \mathbb{N}$ and $0 < \varepsilon < 1$ satisfy $ab = n$, $Q\gamma \ge 2^{a+2}w$, and $\log \binom{Q/2^a}{w} \ge b \ge 2^{a+1}w \log(4/\gamma) + 6a + 2\log(1/\varepsilon)$.

Proof. Let $Q' \stackrel{\mathsf{def}}{=} Q/2^a$, and define $E_{0,k} \stackrel{\mathsf{def}}{=} \left((-1)^{\mathbf{x}_k^\mathsf{T}\mathbf{r}_1}, \cdots, (-1)^{\mathbf{x}_k^\mathsf{T}\mathbf{r}_N}\right)$. We have by the Chernoff bound that $\Pr[|\mathbf{x}_k^\mathsf{T}| > (1/2 - \gamma/4)Q'] \le 2^{-\Omega(Q'\gamma^2)}$. Then, by Lemma 11 with the probability at least $1 - 2^{-\Omega(Q'\gamma^2)}$ and for $\gamma \ge 4w/Q'$

$$\mathsf{bias}(E_{0,k}) \ge N \cdot \left(\frac{2Q'(\gamma/4) - w + 1}{Q' - w + 1}\right)^w \ge N \cdot \left(\frac{\gamma}{2} - \frac{w}{Q'}\right)^w \ge N(\frac{\gamma}{4})^w \ .$$

The condition $\delta^{c^a}\sqrt{N}\varepsilon \geq 2^a c^{2a}$ in Lemma 9 becomes $b \geq 2^{a+1} w \log(4/\gamma) + 6a + 2\log(1/\varepsilon)$, where we set $\delta = (\gamma/4)^w$. The probability argument (and the rest of the proof) is similar Theorem 8 by adding the extra term $2^a \cdot 2^{-\Omega(Q'\gamma^2)}$. □

Table 7. The space, time and sample complexities of different variants of the BKW algorithms for solving the $\mathsf{LPN}_{n,\mu}$ problem with $\mu = (1-\gamma)/2$ and sample complexity $Q = n^{1+\epsilon}$, where $ab = n$, $N_1 = 2^b$, $N_2 = (4/\gamma)^{2^{a+2} \cdot n/(\epsilon \log(n))}$ and constant $\epsilon > 0$ disregarding $\mathsf{poly}(n)$ factors for convenience.

Algorithm	Space	Time	Sample	Condition
Lyubashevsky's [34]	N_1	$N_1 \cdot N_2$	$n^{1+\epsilon}$	$N_1 \approx N_2$
Ours	N_1	N_1	$n^{1+\epsilon}$	$(N_1)^{\log\log(n)} \approx N_2$

As shown in Table 7, we compare [34] with our algorithm for solving $\mathsf{LPN}_{n,\mu}$ problem with $Q = n^{1+\epsilon}$, $\mu = 1/2 - \gamma/2$ and $\gamma \geq 2^{-\log(n)^\sigma}$. Lyubashevsky's technique [34] requires $\log\binom{Q}{w} > n$ to satisfy the entropy condition of the leftover hash lemma, and thus picks $w = 2n/(\epsilon\log(n))$, $a = \kappa \cdot \log\log(n)$ and $b = \frac{n}{\kappa \log\log(n)}$ for positive constants σ, κ satisfying $0 < \kappa + \sigma < 1$. Concretely, consider the extreme case $\gamma = 2^{-\log(n)^\sigma}$ whose running time (omitting $\mathsf{poly}(n)$ factors) $T_{Lyu05}^{n^{1+\epsilon}} \approx 2^b \cdot (1/\gamma)^{2^a \cdot n/\log(n)} \leq 2^{\frac{n}{\kappa\log\log(n)}} \cdot 2^{\frac{n}{\log(n)^{1-\sigma-\kappa}}}$.

In contrast, our algorithm uses all the w-linear combinations and do not require them to look jointly independent, and therefore only need $\log\binom{Q'}{w} \geq b$. As a result, for same values $a = \kappa \cdot \log\log(n)$ and $b = \frac{n}{\kappa\log\log(n)}$, we let $w = 2n/(\epsilon\kappa\log(n)\log\log(n))$ for positive constants κ and σ satisfying $\kappa + \sigma < 1$. One can verify that the three inequalities (for $Q\gamma$, $\log\binom{Q/2^a}{w}$, and b) in Theorem 9 are all satisfied with running time and success probability (where $\varepsilon = 2^{-\log^2 n}$):

$$T_{\text{c-sum}^+\text{bkw}}^{n^{1+\epsilon}} \approx 2^b = 2^{\frac{n}{\kappa\log\log(n)}}$$
$$P_{\text{c-sum}^+\text{bkw}}^{n^{1+\epsilon}} \geq 1 - 2^{6a} \cdot n \cdot \varepsilon - 2^a \cdot 2^{-\Omega(\frac{Q\gamma^2}{2^a})} = 1 - \mathsf{negl}(n) \ .$$

That is, for the same parameter choices our algorithm saves a sub-exponential multiplicative factor $2^{\frac{n}{\log(n)^{1-\sigma-\kappa}}}$ over [34] in running time, where constant $1-\sigma-\kappa$ arbitrarily close to 0 for optimized time complexity. We refer to Table 7 below for a comparison in the general case, which enjoys (for constant $0 < \gamma < 1$) a sub-exponential factor $(4/\gamma)^{2^{a+2} \cdot n/(\epsilon\log(n))}/\mathsf{poly}(n) = 2^{\Omega(n)/\log(n)^{1-\kappa}}$ speedup in running time without consuming (substantially) more space. Note that our N_1 could be even smaller in magnitude than N_2 by using a smaller w and thus produces less stronger noise for majority voting.

Another interesting setting is $\mathsf{LPN}_{n,\mu}$ with $\mu = 1/2 - \gamma/2$, $\gamma \geq 2^{-n^\sigma}$, and $Q = 2^{n^\epsilon}$ for constant $0 < \epsilon < 1$, for which we can keep the time complexity within $2^{O(n/\log(n))}$ as depicted in Table 8. Lyubashevsky's technique [34] picks $w = 2n^{1-\epsilon}$ (to satisfy $\log\binom{Q}{w} > n$), $a = \kappa \cdot \log(n)$ and $b = \frac{n}{\kappa\log(n)}$ for positive

Table 8. The space, time and sample complexities of different variants of the BKW algorithms for solving the $\mathsf{LPN}_{n,\mu}$ problem with $\mu = (1-\gamma)/2$ and sample complexity $Q = 2^{n^\epsilon}$, where $ab = n$, $N_1 = 2^b$, $N_2 = (4/\gamma)^{2^{a+2}\cdot n^{1-\epsilon}}$ and constant $\epsilon > 0$ disregarding $\mathsf{poly}(n)$ factors.

Algorithm	Space	Time	Sample	Condition
Lyubashevsky's [34]	N_1	$N_1 \cdot N_2$	2^{n^ϵ}	$N_1 \approx N_2$
Ours	N_1	N_1	2^{n^ϵ}	$(N_1)^{\log(n)} \approx N_2$

constants σ, κ and ϵ satisfying $\sigma + \kappa < \epsilon$. Concretely, consider the extreme case $\gamma = 2^{-n^\sigma}$ whose running time $T_{Lyu05}^{2^{n^\epsilon}} \approx 2^b \cdot (1/\gamma)^{2^a \cdot n^{1-\epsilon}} \leq 2^{\frac{n}{\kappa \log(n)}} \cdot 2^{n^{1-(\epsilon-\sigma-\kappa)}}$.

In contrast, our algorithm uses the same $a = \kappa \cdot \log(n)$ and $b = \frac{n}{\kappa \log(n)}$ but set $w = 2n^{1-\epsilon}/(\kappa \log(n))$, where positive constants κ, σ and ϵ satisfying $\sigma + \kappa < \epsilon$. This meets all the three conditions (for $Q\gamma$, $\log \binom{Q/2^a}{w}$, and b) in Theorem 9. The resulting running time and success probability (where $\varepsilon = 2^{-\log^2 n}$):

$$T_{\text{c-sum}^+\text{bkw}}^{2^{n^\epsilon}} \approx 2^b = 2^{\frac{n}{\kappa \log(n)}} \quad P_{\text{c-sum}^+\text{bkw}}^{2^{n^\epsilon}} = 1 - \mathsf{negl}(n) \ .$$

That is, for the same parameter choices our algorithm enjoys a sub-exponential factor $2^{n^{1-(\epsilon-\sigma-\kappa)}}$ advantage over [34] in running time, where constant $(\epsilon - \sigma - \kappa)$ is arbitrarily close to 0 for optimized time complexity. We refer to Table 8 below for a comparison in the general case, where for constant $0 < \gamma < 1$ our algorithm saves a sub-exponential factor $(4/\gamma)^{2^{a+2}\cdot n^{1-\epsilon}}/\mathsf{poly}(n) = 2^{O(n^{1-(\epsilon-\kappa)})}$ for arbitrarily small constant $(\epsilon - \kappa)$ with roughly the same space. Note that our N_1 could be even smaller in magnitude than N_2, thanks to the smaller w in use.

Acknowledgements

Yu Yu was supported by the National Key Research and Development Program of China (Grant Nos. 2020YFA0309705 and 2018YFA0704701), the National Natural Science Foundation of China (Grant Nos. 62125204 and 61872236), and the Major Program of Guangdong Basic and Applied Research (Grant No. 2019B030302008). Yu Yu also acknowledges the support from the XPLORER PRIZE and Shanghai Key Laboratory of Privacy-Preserving Computation.

References

1. Albrecht, M.R., Cid, C., Faugère, J., Fitzpatrick, R., Perret, L.: On the complexity of the BKW algorithm on LWE. DCC **74**(2), 325–354 (2015)
2. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography with constant input locality. In: Advances in Cryptology—Crypto 2007. LNCS, vol. 4622, pp. 92–110. Springer (2007)
3. Arratia, R., Goldstein, L., Gordon, L.: Two moments suffice for poisson approximations: the chen-stein method. The Annals of Probability pp. 9–25 (1989)

4. Bai, S., Laarhoven, T., Stehle, D.: Tuple lattice sieving. Cryptology ePrint Archive, Report 2016/713 (2016), https://eprint.iacr.org/2016/713
5. Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic primitives based on hard learning problems. In: Advances in Cryptology—Crypto 1993. pp. 278–291. LNCS, Springer (1994)
6. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. In: 32nd Annual ACM Symposium on Theory of Computing (STOC). pp. 435–440. ACM Press (2000)
7. Bogos, S., Tramèr, F., Vaudenay, S.: On solving LPN using BKW and variants - implementation and analysis. Cryptogr. Commun. **8**(3), 331–369 (2016)
8. Bogos, S., Vaudenay, S.: Optimization of LPN solving algorithms. In: Advances in Cryptology—Asiacrypt 2016, Part I. pp. 703–728. LNCS, Springer (2016)
9. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching (p493-505). Fortschritte Der Physik **46**(4-5) (2010)
10. Budroni, A., Guo, Q., Johansson, T., Mårtensson, E., Wagner, P.S.: Making the BKW algorithm practical for LWE. pp. 417–439. LNCS, Springer (2020)
11. Delaplace, C., Esser, A., May, A.: Improved low-memory subset sum and LPN algorithms via multiple collisions. In: Albrecht, M. (ed.) 17th IMA International Conference on Cryptography and Coding. LNCS, vol. 11929, pp. 178–199. Springer, Oxford, UK (Dec 16–18, 2019)
12. Devadas, S., Ren, L., Xiao, H.: On iterative collision search for LPN and subset sum. pp. 729–746. LNCS, Springer (2017)
13. Dinur, I.: An algorithmic framework for the generalized birthday problem. DCC **87**(8), 1897–1926 (2019)
14. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In: Advances in Cryptology—Crypto 2012. LNCS, vol. 7417, pp. 719–740. Springer (2012)
15. Dohotaru, C., Høyer, P.: Exact quantum lower bound for grover's problem. Quantum Inf. Comput. **9**(5&6), 533–540 (2009)
16. Ducas, L.: Shortest vector from lattice sieving: A few dimensions for free. In: Advances in Cryptology—Eurocrypt 2018, Part I. LNCS, vol. 10820, pp. 125–145. Springer (2018)
17. Esser, A., Heuer, F., Kübler, R., May, A., Sohler, C.: Dissection-BKW. In: Advances in Cryptology—Crypto 2018, Part II. LNCS, vol. 10992, pp. 638–666. Springer (2018)
18. Esser, A., Kübler, R., May, A.: LPN decoded. In: Advances in Cryptology—Crypto 2017, Part II. LNCS, vol. 10402, pp. 486–514. Springer (2017)
19. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: 28th Annual ACM Symposium on Theory of Computing (STOC). pp. 212–219. ACM Press (1996)
20. Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. J. Cryptology **33**(1), 1–33 (Jan 2020)
21. Guo, Q., Johansson, T., Mårtensson, E., Wagner, P.S.: On the asymptotics of solving the LWE problem using coded-BKW with sieving. Cryptology ePrint Archive, Report 2019/009 (2019), https://eprint.iacr.org/2019/009
22. Herold, G., Kirshanova, E.: Improved algorithms for the approximate k-list problem in euclidean norm. In: Intl. Conference on Theory and Practice of Public Key Cryptography 2017, Part I. pp. 16–40. LNCS, Springer (2017)

23. Herold, G., Kirshanova, E., Laarhoven, T.: Speed-ups and time-memory trade-offs for tuple lattice sieving. In: Intl. Conference on Theory and Practice of Public Key Cryptography 2018, Part I. pp. 407–436. LNCS, Springer (2018)
24. Herold, G., Kirshanova, E., May, A.: On the asymptotic complexity of solving LWE. DCC **86**(1), 55–83 (2018)
25. Impagliazzo, R., Zuckerman, D.: How to recycle random bits. In: 30th Annual Symposium on Foundations of Computer Science (FOCS). pp. 248–253. IEEE (1989)
26. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: 15th Annual ACM Symposium on Theory of Computing (STOC). pp. 193–206. ACM Press (1983)
27. Katz, J., Shin, J.S.: Parallel and concurrent security of the HB and HB+ protocols. In: Advances in Cryptology—Eurocrypt 2006. pp. 73–87. LNCS, Springer (2006)
28. Kirchner, P., Fouque, P.A.: An improved BKW algorithm for LWE with applications to cryptography and lattices. In: Advances in Cryptology—Crypto 2015, Part I. LNCS, vol. 9215, pp. 43–62. Springer (2015)
29. Laarhoven, T.: Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In: Advances in Cryptology—Crypto 2015, Part I. LNCS, vol. 9215, pp. 3–22. Springer (2015)
30. Laarhoven, T., Mariano, A.: Progressive lattice sieving. In: PQCrypto 2018. pp. 292–311. Springer (Apr 9–11 2018)
31. Laarhoven, T., de Weger, B.: Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In: Progress in Cryptology—Latincrypt 2015. pp. 101–118. LNCS, Springer (2015)
32. Levieil, É., Fouque, P.A.: An improved LPN algorithm. In: Intl. Conf. on Security and Cryptography for Networks (SCN). pp. 348–359. LNCS, Springer (2006)
33. Liu, H., Yu, Y.: A non-heuristic approach to time-space tradeoffs and optimizations for bkw. Cryptology ePrint Archive, Paper 2021/1343 (2021), https://eprint.iacr.org/2021/1343
34. Lyubashevsky, V.: The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In: RANDOM 2005. pp. 378–389 (2005)
35. Minder, L., Sinclair, A.: The extended k-tree algorithm. J. Cryptology **25**(2), 349–382 (Apr 2012)
36. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: 37th Annual ACM Symposium on Theory of Computing (STOC). pp. 84–93. ACM Press (2005)
37. Schroeppel, R., Shamir, A.: A t=o($2^{n/2}$), s=o($2^{n/4}$) algorithm for certain np-complete problems. SIAM J. Comput. **10**(3), 456–464 (1981)
38. Wagner, D.: A generalized birthday problem. In: Advances in Cryptology—Crypto 2002. LNCS, vol. 2442, pp. 288–303. Springer (2002)