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

Hanlin Liu[1] and Yu Yu*[1]

[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

October 6, 2021

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 $\mu = (1-\gamma)/2$. The BKW solves it with space complexity $2^{\frac{(1+\epsilon)n}{\log n}}$ and time/sample complexity $2^{\frac{(1+\epsilon)n}{\log n}} \cdot 2^{O(n^{\frac{1}{1+\epsilon}})}$ 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/sample complexity $2^{\frac{\log c(1+\epsilon)n}{\log n}} \cdot 2^{O(n^{\frac{1}{1+\epsilon}})}$ and space complexity $2^{\frac{\log c(1+\epsilon)n}{(c-1)\log n}}$, 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 $2^{\frac{\log c(1+\epsilon)n}{\log n}}$, saving factor $2^{\Omega(\frac{n}{1+\epsilon})}$ in time/sample compared to the original BKW, and the variant of Devadas et al. (TCC 2017). This benefits from a careful analysis of the error distribution among the correlated candidates, and therefore avoids repeating the same process $2^{\Omega(\frac{n}{1+\epsilon})}$ times on fresh new samples.
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, 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.

We seek to bridge the gaps between theoretical and heuristic LPN solvers, but take a different approach from Devadas et al. (TCC 2017). We exploit weak yet sufficient conditions (e.g., pairwise independence), and the analysis uses only elementary tools (e.g., Chebyshev's inequality).

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 $\mathbf{a}_i \xleftarrow{\$} \mathbb{F}_2^n$

*E-mail: yuyuathk@gmail.com

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 [BKW03] 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. Run 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. Repeat 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 vote on the m samples obtained in step 2, and produce a candidate for $\mathbf{s}_1$. Repeat 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 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 $T \approx 2^{\frac{(1+\epsilon)n}{\log n}} \cdot 2^{O(n^{\frac{1}{1+\epsilon}})}$, 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 the reality of 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 tacking the above problems followed by a summary of our contributions.

1.2 Time-space Tradeoff for BKW

With the recent advancement of the NIST post-quantum cryptography standardization process, it is increasingly important to give a realistic yet accurate assessment of classic/quantum security strength for the LPN/LWE-based crypto-systems, e.g., by using an automated tool that estimates by extrapolation the concrete security of a crypto-system under specific parameter choices (e.g., [Mar]).

However, the huge memory consumption of BKW has become an obstacle for a realistic security evaluation of LPN/LWE-based crypto-systems. As discussed in [EKM17], while performing 2^{60} or more steps is considered doable with a reasonable budget, an algorithm consuming a RAM of size 2^{60} is out of reach and cannot be instantiated in practice. Likewise, in the lattice setting the enumeration method (e.g., the Kannans algorithm [Kan83] that takes time $2^{O(n \log n)}$ and space $\mathsf{poly}(n)$) often beats the lattice sieving [LM18, Duc18, Laa15, LdW15] (that takes time and space $2^{O(n)}$) in practice, and there is a renewed interest in the time-space trade-offs, e.g., by lattice tuple sieving [BLS16, HK17, HKL18].

Esser et al. [EHK+18] introduced the first variant of the BKW with 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 take 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 steps (repeating the process m times, majority voting, etc.) are similar to the original BKW. Note that c is the parameter that tunes 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 output 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 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 [EHK+18], 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. [EHK+18] 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., [ZJW16, BV16, BTV16]).

Under the independence heuristics, Esser et al. [EHK+18] 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 Fig. 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 [Gro96, DH09, BBHT10]. They also applied the c-sum BKW to the LWE problem [Reg05] and got similar results (see Table 4). We refer to Section 2.4 for more details about the c-sum BKW algorithm. Looking ahead, we provide unconditional versions of all those variants without using any heuristics (see Section 3.2 through Section 3.7).

1.3 Time/sample Optimization and Sample Reduction for BKW

As discussed in Section 1.1, the BKW [BKW03] repeats step 1 for $(1/\gamma)^{2^{a+1}} = 2^{O(n^{\frac{1}{1+\epsilon}})}$ 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 [LF06] presented a heuristic method via fast Walsh-Hadamard transform, often referred to as LF1, that avoids repeating the process many times and recovers multiple secret bits at the same time. Devadas et al. [DRX17] proposed a (non-heuristic) single-list pair-wise

Table 1: The time and space complexities of the c-sum BKW [EHK+18] (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	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}	$N_c^{c-\beta\sqrt{2c}}$	$c = 4, 7, 11, \cdots$ $\beta \in [1, \frac{\sqrt{c}}{\sqrt{c}-1}]$
Quantum	Naive + Grover	N_c	$N_c^{c/2}$	$c \geq 2$

iterative collision search method to optimize the BKW, where they show that the distribution of solutions is close to a Poisson distribution and apply the Chen-Stein method [AGG89] 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.

Lyubashevsky [Lyu05] studied the problem of solving 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. In particular, let $(\mathbf{A}, \mathbf{t}^\mathsf{T} = (\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{x}^\mathsf{T}))$ be the Q 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 statically close to $(\mathbf{A}, \mathbf{U}_n, \mathbf{x}^\mathsf{T}\mathbf{r}_i)$ by the leftover hash lemma [IZ89] 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

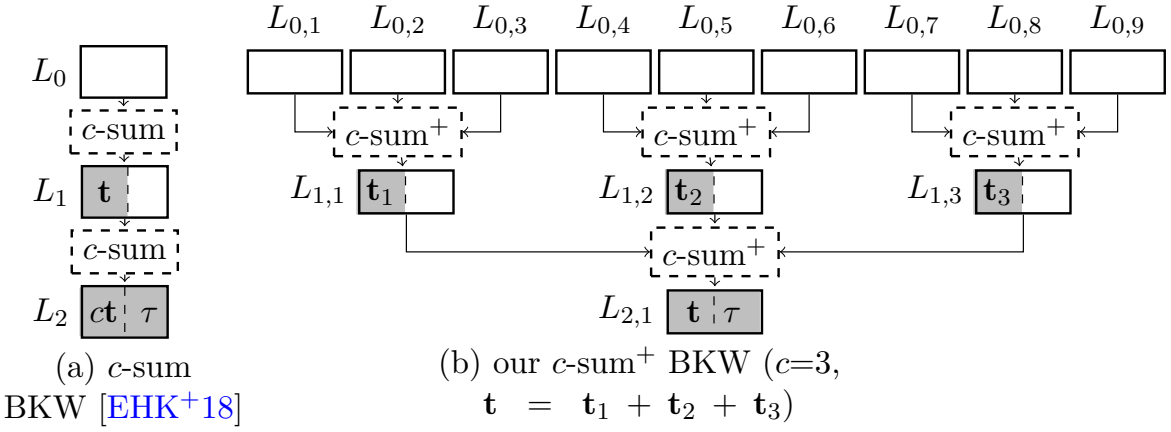
$$T \approx 2^b \cdot (4/\gamma)^{2^{a+2} \cdot n/(\epsilon \log n)}, \quad S \approx 2^b, \quad Q = n^{1+\epsilon} \ .$$

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 $T = 2^{\frac{n}{\kappa \log\log n}} \cdot 2^{O(n)/(\log n)^{1-\kappa}}$, which is optimized when $\kappa \to 1^-$. Let us mention that Lyubashevsky's technique [Lyu05] 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 [Wag02]. We recall that the generalized birthday problem that, given k 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 is a special case for $k = c^a$ ($2 \leq c \in \mathbb{N}$ and $a \in \mathbb{N}$) and we additionally require that the number of solutions found is no less than the size of a single input list (unless they already constitute all solutions). However, unlike Wagner [Wag02], we only require that each list consists of pairwise independent (instead of i.i.d.) uniformly random vectors, and that all the lists are mutually independent.

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 of a much smaller scale, called the c-sum$^+$ problems. More importantly, 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.



(a) c-sum BKW [EHK+18]

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

Figure 1: An illustration of the c-sum BKW [EHK+18] and our c-sum$^+$ BKW.

1. **(Time-space tradeoff).** Our algorithm admits various time-space tradeoffs for solving LPN (shown in Table 1) and LWE (see Table 4), same as those achieved by the c-sum BKW [EHK+18], but without relying on any heuristiscs.

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 $N_2 = (1/\gamma)^{2^{a+1}} = 2^{\Omega(\frac{n}{1+\epsilon})}$ for small constant $\epsilon \to 0^+$ in time and sample complexities compared to the original BKW. 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. [DRX17]. See Table 2 for more details.

3. **(Sample reduction).** By using pairwise independent samples for the initial lists, we provide an alternative to Lyubashevsky's BKW variant [Lyu05] with improved time complexity. In particular, given $Q = n^{1+\epsilon}$ (resp., $Q = 2^{n^\epsilon}$) samples and for constant $0 < \gamma < 1$, 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 [Lyu05]. We refer to Table 3 and Section 4.2 for details.

It might seem counter-intuitive that our results listed in Table 2 and Table 3 only depend on N_1 but still needs to satisfy the condition $N_1 \approx N_2$ (or those in Table 3) for optimized time complexity. As we will see, the condition $N_1 \geq N_2$ (or alike) is translated from that in Theorem 9 to ensure the correctness of majority voting, and we thus let $N_1 \approx N_2$ for optimized complexity and fair comparison.

2 Preliminary

2.1 Notation

We use $\log(\cdot)$ to denote the binary logarithm. For $a \leq b \in \mathbb{N}$, $[a,b] \stackrel{\text{def}}{=} \{a, a+1, \cdots, b\}$ 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

Table 2: 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	N_1	$N_1 \cdot N_2$	$N_1 \cdot N_2$	$N_1 \approx N_2$
Devadas et al.'s [DRX17]	$N_1 \cdot \sqrt{N_2}$	$N_1 \cdot \sqrt{N_2}$	N_1	$N_1 \approx N_2$
Ours	N_1	N_1	N_1	$N_1 \approx N_2$

Table 3: The space, time and sample complexities of different variants of the BKW for solving the $\mathsf{LPN}_{n,\mu}$ problem with $\mu = (1-\gamma)/2$, where $ab = n$, $N_1 = 2^b$, $N_2 = (4/\gamma)^{2^{a+2}\cdot n/(\epsilon \log n)}$ and $N_2' = (4/\gamma)^{2^{a+2}\cdot n^{1-\epsilon}}$.

Sample	Algorithm	Space	Time	Condition
$n^{1+\epsilon}$	Lyubashevskys [Lyu05]	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}	Lyubashevskys [Lyu05]	N_1	$N_1 \cdot N_2'$	$N_1 \approx N_2'$
	Ours	N_1	N_1	$(N_1)^{\log n} \approx N_2'$

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 $\mathbf{s} \xleftarrow{\$} \mathcal{S}$ (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 $1 \le i < j \le N$ the corresponding (l_i, l_j) is uniform over $\mathbb{F}_2^{2b}$.

Lemma 1 (Union Bound)**.** *For all (possibly correlated) events $\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_k$ over a sample space, we have* $\Pr[\mathcal{E}_1 \cup \mathcal{E}_2 \cup \cdots \cup \mathcal{E}_k] \le \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] + \cdots + \Pr[\mathcal{E}_k]$.

Lemma 2 (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 3 (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* $\Pr\left[|X - \mu| \ge \delta\sigma\right] \le \frac{1}{\delta^2}$.

Lemma 4. *For pairwise independent real-valued r.v.s $X_1, \cdots, X_m$ it holds that*

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

We defer the proof of Lemma 4 to Appendix C for completeness.

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 [BFKL93, KS06, AIK09]. 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*[1] $\mathbf{s}$ *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 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 $a \cdot b$ 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 [BKW03])**.** *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 $2^{\frac{(1+\epsilon)n}{\log n} + O(1)n^{\frac{1}{1+\epsilon}}} \approx 2^{\frac{(1+\epsilon+o(1))n}{\log n}}$.

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. [EHK+18] 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 [EHK+18] to work from one iteration to another without losing samples.

Definition 2 (The c-sum Problem (c-SP) [EHK+18])**.** *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* $\mathbf{a}_i \xleftarrow{\$} \mathbb{F}_2^b$ *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. [EHK+18] 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, 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.

We revisit 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

[1] 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.

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** $\mathbf{s} = \mathbf{s}_1 \ldots \mathbf{s}_n$;

independent labels of $\mathbf{u}_1$ 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 [EHK+18]. 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. [EHK+18] introduced the independence heuristic that assumes independence among those vectors. In other words, the performance of the c-sum BKW algorithm was analyzed under the heuristic that these dependencies should only mildly affect the c-sum BKW algorithm. Similar independence heuristics were already used in the optimized analysis of concrete LPN instances [ZJW16, BV16, BTV16]. In the special case of the 2-sum BKW, Devadas et al. [DRX17] proved that these dependencies merely affects the time complexity by an $o(1)$-term in the exponent. Nevertheless, prior to our work there's no formal analysis of the general case $c > 2$, except that some empirical evidences (for certain parameter choices) were provided in [EHK+18].

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

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 $L_1, \cdots, L_c$, where $L_i \stackrel{\mathsf{def}}{=} (\mathbf{a}_{i,1}, \cdots, \mathbf{a}_{i,N}) \in \mathbb{F}_2^{b \cdot N}$, be c lists such that*

1. ***(Intra-list pairwise independence).*** *Within each list L_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 over $\mathbb{F}_2^b \times \mathbb{F}_2^b$.*

2. ***(Inter-list independence).*** *$L_1, \cdots, L_c$, each seen as a random variable, are all mutually independent.*

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 [EHK+18]. As we will see, $2/N = \mathsf{negl}(n)$ for a super-polynomial N already suffices.

Lemma 5. *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 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,

$$\begin{aligned}\Pr\Big[X < N \text{ or } X > 3N\Big] &\le \Pr\Big[|X - \mathbb{E}[X]| > N\Big] \\ &\le \frac{Var[X]}{N^2} = \frac{\sum_S Var[X_S]}{N^2} \le \frac{\mathbb{E}[X]}{N^2} = \frac{2}{N} \quad ,\end{aligned}$$

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 second equality is due to Lemma 4, and the last inequality is due to

$$Var[X_i] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 \le \mathbb{E}[X_i^2] = \mathbb{E}[X_i] \quad .$$

□

3.2 The c-sum$^+$ BKW

We introduced the c-sum$^+$ problem in Definition 3, and we show in Lemma 5 that it has at least N solutions (except with 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, 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.,

$$\left\{ \bigoplus_{i=1}^{c} \mathbf{a}_{i,k_i}^{j+1} \middle| (k_1, \cdots, k_c) \in [N]^c, \bigoplus_{i=1}^{c} \mathbf{a}_{i,k_i}^{j} = \mathbf{t} \right\} \quad ,$$

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.

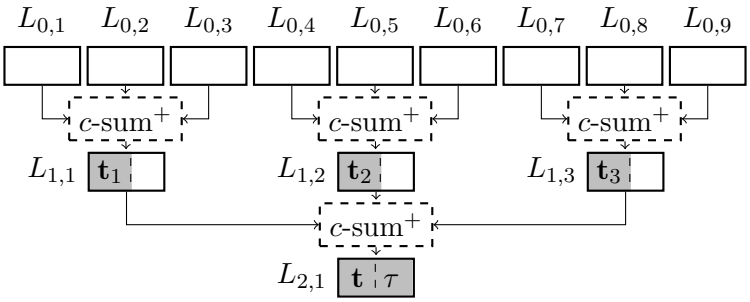


Figure 2: An illustration of the c-sum$^+$ BKW for $c = 3$, where $\mathbf{t} = \mathbf{t}_1 + \mathbf{t}_2 + \mathbf{t}_3$

Lemma 6 (Pairwise independence preserving)**.** *If the leaf-level lists $L_{0,1}, \ldots, L_{0,c^a}$ 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 $L_{j,1}, \ldots, L_{j,c^{a-j}}$ 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 $L_{j,1}, \ldots, L_{j,c^{a-j}}$ 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 6). 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).

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** $\mathbf{s} = \mathbf{s}_1 \ldots \mathbf{s}_n$;

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 probability P as below*

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

where $T_{c,N,b}$ 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, $ab \geq n$, and $N = 2^{\frac{b+1}{c-1}}$ for $2 \leq c \in \mathbb{N}$.

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 5, the c-sum$^+$ problem (implicitly defined 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 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 (see Lemma 1).

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 2). 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

$$M \approx M_{c,N,b} \cdot c^a \ , \quad T \approx T_{c,N,b} \cdot c^a \cdot (\frac{1}{\gamma})^{2 \cdot c^a} \ .$$

□

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 Algorithm 3. Similar to the naive approach [EHK+18], 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 $\mathbf{t} \in \mathbb{F}_2^b$.

Algorithm 3: Naive c-sum$^+$

Input: $L_1, \cdots, L_c \in (\mathbb{F}_2^b)^N$ and $\mathbf{t} \in \mathbb{F}_2^b$, where $L_j \stackrel{\mathsf{def}}{=} (\mathbf{a}_{j,1}, \cdots, \mathbf{a}_{j,N})$ for $j \in [c]$
Output: $\mathcal{S} \subset \binom{[N]}{c}$ or $\perp$

```
Sort out L_c ;
for all V = (i_1, ..., i_{c-1}) ∈ [N]^{c-1} do
    p := ⊕_{j∈[c-1]} a_{j,i_j};
    for all i_c ∈ [N] satisfying a_{c,i_c} = t ⊕ p do
        S ← S ∪ {{i_1, ..., i_c}};
        if |S| = N then
            Return S;
Return ⊥;
```

Lemma 7. *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 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. □

We obtain Theorem 3 by combining Lemma 7 with Theorem 2.

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 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 constant noise $\mu = 1/4$, we set $a = \frac{\log n}{\log c(1+\epsilon)}$ and $b = \frac{\log c(1+\epsilon) n}{\log n}$ to get

$$\log M = \frac{\log c}{c-1} \cdot \frac{n(1+\epsilon)}{\log n}, \quad \log T = \log c \cdot \frac{n(1+\epsilon+o(1))}{\log n}, \quad P \geq 1 - \mathsf{negl}(n).$$

3.4 Quantum c-sum$^+$ BKW Algorithm

Following the steps in [EHK+18], we adopt the Grovers algorithm [Gro96] (see Theorem 4) to quantumly speed up the crucial (and time-consuming) first step in the naive c-sum$^+$. 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.

Theorem 4 (Grover Algorithm [Gro96,DH09,BBHT10]). *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*

$$q = \tilde{O}\Big(\sqrt{\frac{|D|}{|f^{-1}(1)|}}\Big) \quad .$$

Algorithm 4: Quantum c-sum$^+$

Input: $L_1, \cdots, L_c \in (\mathbb{F}_2^b)^N$ and $\mathbf{t} \in \mathbb{F}_2^b$, where $L_j \stackrel{\mathsf{def}}{=} (\mathbf{a}_{j,1}, \cdots, \mathbf{a}_{j,N})$ for $j \in [c]$
Output: $\mathcal{S} \subset \binom{[N]}{c}$ or $\perp$

```
Sort out L_c ;
for repeat Õ(N) times do
    (i_1, ··· , i_{c-1}) ← Grover^{f_t} ;
    p := ⊕_{j∈[c-1]} a_{j,i_j};
    for all i_c ∈ [N] satisfying a_{c,i_c} = t ⊕ p do
        S ← S ∪ {(i_1, ··· , i_c)};
        if |S| = N then
            Return S;
Return ⊥;
```

Lemma 8 follows from Theorem 4 and Lemma 5.

Lemma 8. *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 probability $1 - 2/N$.*

Combining Lemma 8 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 $M \approx N \cdot c^a$ with 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 $N = 2^{\frac{b+1}{c-1}}$.*

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

$$\log M = \frac{\log c}{c-1} \cdot \frac{n(1+\epsilon)}{\log n} \quad , \quad \log T = \frac{c \cdot \log c}{2(c-1)} \cdot \frac{n(1+\epsilon+o(1))}{\log n} \quad , \quad P \geq 1 - \mathsf{negl}(n),$$

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. [EHK+18] borrowed the dissection technique from [SS81,DDKS12] 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 [EHK+18] 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.

Definition 4 (Join Operator [EHK+18])**.** *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 $d' \le d$, the join of L_1 and L_2 on target $\mathbf{t}$ is defined as

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

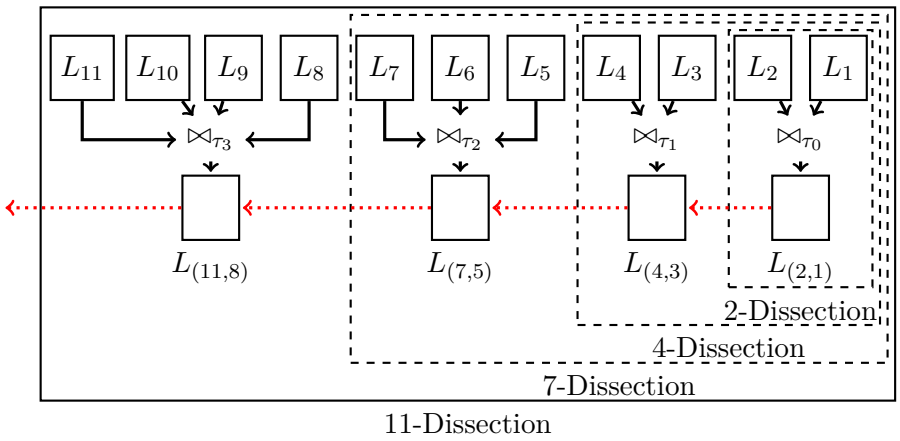


Figure 3: An illustration of the dissection 11-sum on input lists $L_{11}, \cdots, L_1$ that recursively invokes dissection 7- and 4-sum (in dashed boxes), where $\bowtie_\tau$ is the join operator (as per Definition 4) and implemented by Naive c-sum$^+$ (as per Algorithm 3), the blank box stores the 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 5 (The Magic Sequence [DDKS12])**.** *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 $L_{(c_i, c_{i-1}+1)\cdot}$

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 Algorithm 7 and Algorithm 8) 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 see Lemma 17), and the overall space consumption is $O(iN)$.

The dissection c_i-sum$^+$ (Algorithm 7) invokes the interative procedure c_j-Dissect for $j \le i$ (Algorithm 8) to solve the c_i-sum$^+$ problem for $c_i \in \mathsf{magic}$. We already show in Lemma 5 that for any $2 \le c \in \mathbb{N}$ the problem has at least N solutions (except with probability $2/N$). Esser et al. [EHK+18] showed that the dissection c_i-sum$^+$ does an exhaustive search over all solutions.

Lemma 9 (Correctness of c_i-Dissect [EHK+18])**.** *For some fixed k_j, let $\mathbf{a}_{j,k_j} := L_j(k_j)$ denote the k_j-th element of the list L_j. For every $c_i \in \mathsf{magic}$, when the c_i-Dissect$(L_{c_i}, \cdots, L_1, \mathbf{t}, outer)$ (see Algorithm 8) halts, the set $\mathcal{S}$ contains $(k_{c_i}, \cdots, k_1) \in [2^\lambda]^{c_i}$ if and only if $\bigoplus_{j=1}^{c_i} \mathbf{a}_{j,k_j} = \mathbf{t}$.*

Compared with the naive c-sum$^+$ algorithm that also exhausts all solutions, dissection c_i-sum$^+$ enjoys optimized time complexity as stated in Lemma 10. Esser et al. [EHK+18] analyzed the c_i-Dissect subroutine (essentially the $\bowtie_{\tau_j}$ operation in Fig. 3, see also Algorithm 8) in terms of expected time and space, and we further give their upper bounds in Lemma 18 and Lemma 20 to reach a more formal statement in Lemma 10

Lemma 10. *For every $c_i \in \mathsf{magic}$, the Dissection c_i-sum$^+$ algorithm solves the c_i-sum$^+$ problem with the target length b and list size $N \ge 2^{\frac{b+1}{c_i-1}}$ in time $T \approx N^{c_{i-1}}$ and space $M \approx N$, and it returns N distinct solutions with the probability at least $1 - O(i)/N$.*

Proof. Based on Lemma 5, there are at least N distinct solutions with the probability at least $1 - 2/N$ that will be exhaustively recovered by Lemma 9. Note that $\lambda = \frac{b+1}{c_i-1} \le \frac{b}{i}$ as required

by the c_i-Dissect. Based on Lemma 18 and Lemma 20, we have

$$\Pr[M \le N \cdot \mathsf{poly}(b, c_i)] \ge 1 - O(i)/N \ , \ \Pr[T \le N^{c_{i-1}} \cdot \mathsf{poly}(b, c_i)] \ge 1 - O(i)/N \ .$$

Therefore, the Dissection c_i-sum$^+$ algorithm finds N distinct solutions in time $T \approx N^{c_{i-1}}$ and space $M \approx N$ with the probability at least $1 - O(i)/N$ via the union bound (see Lemma 1). □

Combining Lemma 10 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 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 $N = 2^{\frac{b+1}{c-1}}$.*

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 $\frac{i}{c_i - 1}$ factor represents the optimization over the naive c-sum$^+$ BKW.

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. [EHK+18] 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 [EHK+18] (see 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 Algorithm 8) 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 Algorithm 8). Instead of relying on any heuristics, we prove in Lemma 11 unconditionally that the outmost for-loop needs only 2^y iterations for $y = b - c_{i-1} \cdot \lambda + 1$. We defer the proofs of Lemma 11 and Lemma 12 to Appendix B due to the similarities to Lemma 5 and Lemma 10 respectively. Combining Lemma 12 and Theorem 2, we obtain Theorem 7.

Algorithm 5: Tailored Dissection c_i-sum$^+$

Input: $L_{c_i}, \cdots, L_1 \in (\mathbb{F}_2^b)^{N^\beta}$ and the target $\mathbf{t} \in \mathbb{F}_2^b$
Output: $\mathcal{S} \subset \binom{[N^\beta]}{c_i}$ or $\perp$
1 $\mathcal{S} \leftarrow c_i$-Dissect$(L_{c_i}, \cdots, L_1, \mathbf{t}, \text{outer})$ // halt c_i-Dissect once $|S| = N^\beta$;
2 **if** $|\mathcal{S}| < N^\beta$ **then**
3 **Return** $\perp$;
4 **Return** $\mathcal{S}$;

Lemma 11. *For every $c_i \in \mathsf{magic}$, the first 2^y-th iterations of the outmost loop (see line 1) of the c_i-Dissect (see Algorithm 8) has at least N^β distinct solutions and at most $3 \cdot N^\beta$ distinct solutions with probability $1 - O(1)/N^\beta$, where $N = 2^{\frac{b+1}{c_i - 1}}$, $\beta \in [1, \frac{c_i - 1}{c_{i-1}}]$, $\lambda = \frac{\beta \cdot (b+1)}{c_i - 1}$ and $y = b - c_{i-1} \cdot \lambda + 1$.*

Lemma 12. *For every $c_i \in \mathsf{magic}$, the tailored Dissection c_i-sum$^+$ algorithm solves the c_i-sum$^+$ problem with the target length b and list size $N^\beta \geq 2^{\frac{\beta\cdot(b+1)}{c_i-1}}$ (for $\beta \in [1, \frac{c_i-1}{c_{i-1}}]$) in time $T \approx N^{c_{i-1}-(\beta-1)\cdot i}$ and space $M \approx N^\beta$, and it returns N^β distinct solutions with the probability at least $1 - O(i)/N^\beta$.*

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 $T \approx N^{c_{i-1}-(\beta-1)\cdot i} \cdot c_i^a \cdot (\frac{1}{\gamma})^{2\cdot c_i^a}$ and space $M \approx N^\beta \cdot c_i^a$ with 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

$$\log M = \frac{\beta \cdot \log c_i}{c_i - 1} \cdot \frac{n(1+\epsilon)}{\log n},\ \log T = (1 - \frac{\beta \cdot i}{c_i - 1}) \cdot \log c_i \cdot \frac{n(1+\epsilon+o(1))}{\log n},\ P \geq 1 - \mathsf{negl}(n),$$

where the difference to the dissection c-sum$^+$ BKW was highlighted.

3.7 Time-space Trade-offs for solving LWE

Regev [Reg05] 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. [ACF+15] adapted the BKW algorithm to solve the LWE problem. 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 [KF15]. Herold et al. [HKM18] 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. [EHK+18], 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 [HKM18], 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 4, which are essentially the same as that of the c-sum BKW for LWE [EHK+18] except that our c-sum$^+$ BKW does not rely on any heuristics.

4 Optimization and Sample Reduction for BKW

We adjust our dissection 2-sum$^+$ BKW to optimize the time, space and sample complexities of the original BKW algorithm. Moreover, it can further push the sample complexity to 2^{n^ϵ} and even $n^{1+\epsilon}$, which also optimize the complexities over those achieved using Lyubashevskys technique [Lyu05].

Table 4: 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	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	$N_c^{c/2}$	$c \geq 2$

Table 5: 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	N_1	$N_1 \cdot N_2$	$N_1 \cdot N_2$	$N_1 \approx N_2$
Devadas et al.'s [DRX17]	$N_1 \cdot \sqrt{N_2}$	$N_1 \cdot \sqrt{N_2}$	N_1	$N_1 \approx N_2$
Our 2-sum$^+$ BKW 2.0	N_1	N_1	N_1	$N_1 \approx N_2$

4.1 Time, Space, and Sample Optimizations

As shown in Table 5, we compare the results of the original BKW, Devadas et al.'s optimized version [DRX17] and our 2-sum$^+$ BKW 2.0 (stated in Theorem 8).

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 $ab = n$. Our 2-sum$^+$ BKW 2.0 requires essentially the same condition, i.e., $b = 2^{a+1}\log(1/\gamma) + O(\log n)$ (as specified in Theorem 8). 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 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. Devadas et al. [DRX17] further 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 2.0 enjoys a sub-exponential factor advantage both in time/space complexities compared to [DRX17].

Majority voting on correlated samples. The c-sum BKW [EHK+18] 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 $2^{O(n^{\frac{1}{1+\epsilon}})}$. Instead, we recover the single bit of secret via a majority voting on the elements in $L_{a,1}$ (line 7 in Algorithm 6). This is non-trivial since the 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[2]. 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 $L_{j,k}$ 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 difference[3] between the numbers of

[2]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 $0 < \mu < 0.5$.

[3]Here the formula of $\mathsf{bias}(E_{j,k})$ is informal, and only serves to explain the necessary and sufficient conditions for a successful majority voting.

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 2.0 We now describe how to adapt the c-sum$^+$ BKW (Algorithm 6) 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$^+$ (Algorithm 3) that halt as soon as N solutions are found. In contrast, we let the c-sum$^+$ BKW 2.0 be exhaustive, i.e., the underlying c-sum$^+$ problem solver (see 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 14, 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 probability 2^{-b}. Further, whether an element in $JE_{j,1}$ hits the specified target or not is a pairwise independent event (see Lemma 6). 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 14, and prior to that we introduce Lemma 13 as an analogue of the piling-up lemma that characterizes how the bias is amplified through the c-sum$^+$ operations

Algorithm 6: The c-sum$^+$ BKW 2.0

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

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

Lemma 13. *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 14. *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 $E_{0,1}, \ldots, E_{0,c^a}$ 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., $\mathsf{bias}(E_{0,k}) \ge \delta N$ and $|E_{0,k}| = N$ for every $1 \le k \le c^a$. 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 probability at least $1 - c^{4j+1}\varepsilon$, we have by the definition of $\bowtie$

$$N^c(1 - \frac{2^j c^{2j+1}}{\sqrt{N}\varepsilon}) \le N^c(1 - \frac{2^j c^{2j}}{\sqrt{N}\varepsilon})^c < |JE_{j+1,1}| < N^c(1 + \frac{2^j c^{2j}}{\sqrt{N}\varepsilon})^c \le N^c(1 + \frac{2^{j+1} c^{2j+1}}{\sqrt{N}\varepsilon})$$

where by Lemma 22 $(1+d)^c \le 1+2cd$ and $(1-d)^c \ge 1-cd$ for $0 < cd < 1$, $c \ge 2$. 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 6). Thus, the above implies (recall $N^{c-1} = 2^b$)

$$\Pr\Big[\big|\mathbb{E}\big[|E_{j+1,1}|\big] - N\big| < \frac{2^{j+1}\sqrt{N}c^{2j+1}}{\varepsilon}\Big] \ge 1 - c^{4j+1}\varepsilon \ .$$

Similar to the proof of Lemma 5 (except for a different value of N), we have

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

By Lemma 13 the following holds with probability at least $1 - c^{4j+1}\varepsilon$

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

where the Bernoulli's inequality $(1-d)^c \ge 1 - cd$ is applicable since $c \ge 2$ and $d = \frac{2^j c^{2j}}{\delta^{c^j}\sqrt{N}\varepsilon} < \frac{2^a c^{2a}}{\delta^{c^a}\sqrt{N}\varepsilon} \le 1$. 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 6 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

$$\Pr\Big[\mathbb{E}\big[\mathsf{bias}(E_{j+1,1})\big] > \delta^{c^{j+1}} N - \frac{2^j\sqrt{N}c^{2j+1}}{\varepsilon}\Big] \ge 1 - c^{4j+1}\varepsilon \ ,$$

and thus

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

where the analysis is essential the same as that for bounding $|E_{j+1,1}|$ except that

$$Var\big[\mathsf{bias}(E_{j+1,1})\big] = \sum_{l=1}^{|JE_{j+1,1}|} Var\big[\mathbf{v}_l\cdot(-1)^{e_l}\big] \le \sum_{l=1}^{|JE_{j+1,1}|} \mathbb{E}\big[\mathbf{v}_l\big] = \mathbb{E}\Big[\sum_{l=1}^{|JE_{j+1,1}|}\mathbf{v}_l\Big] \ .$$

□

We state the optimized algorithm for $c = 2$ in Theorem 8, and compare it with the original BKW and the one by Devadas et al. [DRX17] in Table 5.

Theorem 8 (The 2-sum$^+$ BKW 2.0)**.** *The* $\mathsf{LPN}_{n,\mu}$ *problem with* $\mu = 1/2 - \gamma/2$ *and* $\gamma > 2^{-b/3}$ *can be solved in time* T*, space* M *with probability* P *as below*

$$T \approx 2^{a+b} \ , \quad M \approx 2^{a+b} \ , \quad P \ge 1 - 2^{6a}\cdot n\cdot\varepsilon \ ,$$

where $ab = n$*,* $b > n^{0.6}$*, and* $b \ge 2^{a+1}\log(1/\gamma) + 6a + 2\log(1/\varepsilon) + \mathsf{negl}(n)$*.*

Proof. Set the δ in Lemma 14 to $\gamma - 2^{-\frac{b}{2}}\sqrt{\log(1/\varepsilon)}$, and we have by Chernoff bound

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

where $N = 2^b$ for $c = 2$. The condition $\delta^{c^a}\sqrt{N}\varepsilon \ge 2^a c^{2a}$ in Lemma 14 is now

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

By Lemma 14 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 $1 - 2^{5a}\cdot n\cdot\varepsilon$. Therefore, the 2-sum$^+$ algorithm (see Algorithm 9) exhaustively find all solutions of each 2-sum$^+$ problem instance in time and space complexities at most $2^b\cdot\mathsf{poly}(n)$ with probability at least $1 - 2^{5a}\cdot n\cdot\varepsilon$. As for the correctness, the bias of the final list $E_{a,1}$ is positive with the probability at least $1 - 2^{4a}\cdot\varepsilon$ in order to successfully recover a single bit of the secret. Overall, it recovers the whole secret correctly with probability more than $1 - 2^{4a}\cdot n\cdot\varepsilon$ by the union bound. □

4.2 Sample Reduction for BKW

Lyubashevsky [Lyu05] 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 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, and generate 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 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 statically close to $(\mathbf{A},\ \mathbf{U}_n,\ \mathbf{x}^\mathsf{T}\mathbf{r}_i)$ by the leftover hash lemma [IZ89], which requires min-entropy $\mathbf{H}_\infty(\mathbf{r}_i) = \log\binom{Q}{w} > n$. Therefore, Lyubashevsky [Lyu05] 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 2.0 (Algorithm 6) initialize the lists $L_{0,1}, \ldots, L_{0,c^a}$, with independent fresh LPN samples. However, the pairwise independence preserving lemma (Lemma 6) 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 load 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 2.0 (line 2 of Algorithm 6), which takes time and space 2^{a+b}. The rest steps are the same as those in Algorithm 6.

Lemma 15. *For $k = o(m)$ we have $\log\binom{m}{k} = (1+o(1))k\log\frac{m}{k}$.*

Lemma 16 ([Lyu05])**.** *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 2.0 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 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 15 with 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 $\delta^{c^a}\sqrt{N}\varepsilon \ge 2^a c^{2a}$ in Lemma 14 becomes $b \ge 2^{a+1}w\log(4/\gamma) + 6a + 2\log(1/\varepsilon)$, where we set $\delta = (\gamma/4)^w$. The probability argument (and the rest of the proof) is similar to that of Theorem 8 by adding the extra term $2^a\cdot 2^{-\Omega(Q'\gamma^2)}$. □

As shown in Table 6, we compare [Lyu05] with our algorithm for solving $\mathsf{LPN}_{n,\mu}$ problem with $Q = n^{1+\epsilon}$, $\mu = 1/2 - \gamma/2$ and $\gamma \ge 2^{-(\log n)^\sigma}$. Lyubashevsky's technique [Lyu05] 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)

$$T_{Lyu05}^{n^{1+\epsilon}} \approx 2^b\cdot(1/\gamma)^{2^a\cdot n/\log n} \le 2^{\frac{n}{\kappa\log\log n}}\cdot 2^{\frac{n}{(\log n)^{1-\sigma-\kappa}}}\ .$$

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 $\varepsilon = 2^{-\log^2 n}$):

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

That is, for the same parameter choices our algorithm saves a sub-exponential multiplicative factor $2^{\frac{n}{(\log n)^{1-\sigma-\kappa}}}$ over [Lyu05] in running time, where constant $1 - \sigma - \kappa$ arbitrarily close to 0 for optimized time complexity. We refer to Table 6 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.

Table 6: The space, time and sample complexities of different variants of the BKW algorithms for solving the $\mathsf{LPN}_{n,\mu}$ problem with $\mu = (1-\gamma)/2$ and sample complexity $Q = n^{1+\epsilon}$, where $ab = n$, $N_1 = 2^b$ and $N_2 = (4/\gamma)^{2^{a+2} \cdot n/(\epsilon \log n)}$ disregarding $\mathsf{poly}(n)$ factors for convenience.

Algorithm	Space	Time	Sample	Condition
Lyubashevskys [Lyu05]	N_1	$N_1 \cdot N_2$	$n^{1+\epsilon}$	$N_1 \approx N_2$
Ours	N_1	N_1	$n^{1+\epsilon}$	$(N_1)^{\log\log n} \approx N_2$

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 7. Lyubashevsky's technique [Lyu05] 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 constants σ, κ and ϵ satisfying $\sigma + \kappa < \epsilon$. Concretely, consider the extreme case $\gamma = 2^{-n^\sigma}$ whose running time

$$T^{2^{n^\epsilon}}_{Lyu05} \approx 2^b \cdot (1/\gamma)^{2^a \cdot n^{1-\epsilon}} \leq 2^{\frac{n}{\kappa \log n}} \cdot 2^{n^{1-(\epsilon-\sigma-\kappa)}}$$

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 $\varepsilon = 2^{-\log^2 n}$):

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

That is, for the same parameter choices our algorithm enjoys a sub-exponential factor $2^{n^{1-(\epsilon-\sigma-\kappa)}}$ advantage over [Lyu05] in running time, where constant $(\epsilon - \sigma - \kappa)$ is arbitrarily close to 0 for optimized time complexity. We refer to Table 7 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.

References

[ACF+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Des. Codes Cryptogr.*, 74(2):325–354, 2015.

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 $Q = 2^{n^\epsilon}$, where $ab = n$, $N_1 = 2^b$ and $N_2 = (4/\gamma)^{2^{a+2}\cdot n^{1-\epsilon}}$ disregarding $\mathsf{poly}(n)$ factors.

Algorithm	Space	Time	Sample	Condition
Lyubashevskys [Lyu05]	N_1	$N_1 \cdot N_2$	2^{n^ϵ}	$N_1 \approx N_2$
Ours	N_1	N_1	2^{n^ϵ}	$(N_1)^{\log n} \approx N_2$

[AGG89] Richard Arratia, Larry Goldstein, and Louis Gordon. Two moments suffice for poisson approximations: the chen-stein method. *The Annals of Probability*, pages 9–25, 1989.

[AIK09] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. *J. Cryptol.*, 22(4):429–469, 2009.

[BBHT10] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching (p493-505). *Fortschritte Der Physik*, 46(4-5):–, 2010.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *Advances in Cryptology - CRYPTO '93*, pages 278–291. Springer, 1993.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *IACR Cryptol. ePrint Arch.*, 2016:713, 2016.

[BTV16] Sonia Bogos, Florian Tramèr, and Serge Vaudenay. On solving L P N using B K W and variants - implementation and analysis. *Cryptogr. Commun.*, 8(3):331–369, 2016.

[BV16] Sonia Bogos and Serge Vaudenay. Optimization of LPN solving algorithms. In *Advances in Cryptology - ASIACRYPT 2016*, pages 703–728, 2016.

[DDKS12] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In *Advances in Cryptology - CRYPTO 2012*, pages 719–740. Springer, 2012.

[DH09] Catalin Dohotaru and Peter Høyer. Exact quantum lower bound for grover's problem. *Quantum Inf. Comput.*, 9(5&6):533–540, 2009.

[DRX17] Srinivas Devadas, Ling Ren, and Hanshen Xiao. On iterative collision search for LPN and subset sum. In *Theory of Cryptography Conference, TCC 2017*, pages 729–746. Springer, 2017.

[Duc18] Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In *Advances in Cryptology - EUROCRYPT 2018*, pages 125–145, 2018.

[EHK+18] Andre Esser, Felix Heuer, Robert Kübler, Alexander May, and Christian Sohler. Dissection-bkw. In *Advances in Cryptology - CRYPTO 2018*, pages 638–666, 2018.

[EKM17] Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In *Advances in Cryptology - CRYPTO 2017*, pages 486–514. Springer, 2017.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing 1996*, pages 212–219. ACM, 1996.

[HK17] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in euclidean norm. In *Public-Key Cryptography - PKC 2017*, pages 16–40. Springer, 2017.

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time-memory trade-offs for tuple lattice sieving. In *Public-Key Cryptography - PKC 2018*, pages 407–436. Springer, 2018.

[HKM18] Gottfried Herold, Elena Kirshanova, and Alexander May. On the asymptotic complexity of solving LWE. *Des. Codes Cryptogr.*, 86(1):55–83, 2018.

[IZ89] Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *30th Annual Symposium on Foundations of Computer Science*, pages 248–253, 1989.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 193–206. ACM, 1983.

[KF15] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In *Advances in Cryptology - CRYPTO 2015*, pages 43–62. Springer, 2015.

[KS06] Jonathan Katz and Ji Sun Shin. Parallel and concurrent security of the HB and hb$^+$ protocols. In *Advances in Cryptology - EUROCRYPT 2006*, pages 73–87. Springer, 2006.

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Advances in Cryptology - CRYPTO 2015*, pages 3–22. Springer, 2015.

[LdW15] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In *Progress in Cryptology - LATINCRYPT 2015*, pages 101–118. Springer, 2015.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In *Security and Cryptography for Networks, SCN 2006*, pages 348–359. Springer, 2006.

[LM18] Thijs Laarhoven and Artur Mariano. Progressive lattice sieving. In *Post-Quantum Cryptography, PQCrypto 2018*, pages 292–311. Springer, 2018.

[Lyu05] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Proceedings of 9th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM 2005)*, pages 378–389, 2005.

[Mar] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn Postlethwaite, Fernando Virdia, Thomas Wunderer. Estimate all the NTRU/LWE schemes. https://estimate-all-the-lwe-ntru-schemes.github.io/docs/. Online; accessed 30 January 2020.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM, 2005.

[SS81] Richard Schroeppel and Adi Shamir. 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.

[Wag02] David A. Wagner. A generalized birthday problem. In *Advances in Cryptology - CRYPTO 2002*, pages 288–303. Springer, 2002.

[ZJW16] Bin Zhang, Lin Jiao, and Mingsheng Wang. Faster algorithms for solving LPN. In *Advances in Cryptology - EUROCRYPT 2016*, pages 168–195. Springer, 2016.

Supplementary Material

A The Dissection c-sum$^+$ BKW in Details

The dissection c_i-sum$^+$ (Algorithm 7) invokes the interative procedure c_j-Dissect for $j \le i$ (Algorithm 8) to fulfill the task, where "inner" indicates that the c_i-Dissect was called recursively by the c_{i+1}-Dissect, and "outer" means that the c_i-Dissect was called by the Dissection c_i-sum.

Algorithm 7: Dissection c_i-sum Algorithm

Input: $L_{c_i}, \cdots, L_1 \in (\mathbb{F}_2^b)^N$ and $\mathbf{t} \in \mathbb{F}_2^b$
Output: $\mathcal{S} \subset [N]^{c_i}$ or $\perp$

$\mathcal{S} \leftarrow c_i$-Dissect$(L_{c_i}, \cdots, L_1, \mathbf{t}, \text{outer})$;
if $|\mathcal{S}| < N$ **then**
  Return $\perp$;
Return $\mathcal{S}$;

Algorithm 8: c_i-Dissect$(L_{c_i}, \cdots, L_1, \mathbf{t}, \text{pos})$, where $c_i \in \mathsf{magic}$

Input: $L_{c_i}, \cdots, L_1 \in (\mathbb{F}_2^b)^{2^\lambda}$ where $\lambda \le \frac{b}{i}$, $\mathbf{t} \in \mathbb{F}_2^b$ and pos $\in$ {inner, outer}
Output: $\mathcal{S} \subset \binom{[N]}{c_i}$ or $\perp$

1 **for** *all* $\tau_i \in \mathbb{F}_2^{i\cdot\lambda}$ **do**
2 $L_{(c_i, c_{i-1}+1)} \leftarrow L_{c_i} \bowtie_{\tau_i} (L_{c_i-1} \bowtie \cdots \bowtie L_{c_{i-1}+1})$;
3 **for** *all* $\mathbf{a}_{(c_{i-1},1)}$ *passed from* c_{i-1}*-Dissect*$(L_{c_{i-1}}, \cdots, L_1, \mathsf{low}_{i\cdot\lambda}(\mathbf{t}) \oplus \tau_i, \textit{inner})$ **do**
4 **for** *all* $\mathbf{a}_{(c_i,1)} \in L_{(c_i,c_{i-1}+1)} \bowtie_{\mathbf{t}} \mathbf{a}_{c_{i-1},1}$ **do**
5 **if** *pos* $=$ *inner* **then**
6 pass $\mathbf{a}_{(c_i,1)}$ to c_{i+1}-Dissect;
7 **else**
8 $\mathcal{S} \leftarrow \mathcal{S} \cup \{\text{recover indices}(\mathbf{a}_{(c_i,1)})\}$;
9 **if** $|\mathcal{S}| \ge 2^\lambda$ **then**
10 **Return** $\mathcal{S}$;
11 **Return** $\perp$;

Proof of Lemma 9. We prove it by induction.

1. The base clause $i = 1$. Assume $(k_4, k_3, k_2, k_1) \in [2^\lambda]^4$ is a single-solution, i.e., there exists τ, such that
$$\mathbf{a}_{4,k_4} \oplus \mathbf{a}_{3,k_3} = \tau \text{ and } \mathbf{a}_{2,k_2} \oplus \mathbf{a}_{1,k_1} = \mathbf{t} \oplus \tau \ .$$
Therefore, we will store $\mathbf{a}_{4,k_4} \oplus \mathbf{a}_{3,k_3}$ in $L_{4,3}$ (see line 2) if $\tau_1 := \mathsf{low}_\lambda(\tau)$ fixed in line 1. At the same time, we will pick $\mathbf{a}_{2,k_2} \oplus \mathbf{a}_{1,k_1}$ in line 3 since $\mathbf{a}_{2,k_2} \oplus \mathbf{a}_{1,k_1} \in L_2 \bowtie_{\mathsf{low}_\lambda(\mathbf{t})\oplus\tau_1} L_1$. Therefore, the solution (k_4, k_3, k_2, k_1) is recovered in line 4 and added to $\mathcal{S}$ in line 8. If $(k_4, k_3, k_2, k_1) \in [2^\lambda]^4$ is not a single-solution, then for some τ, we have
$$\mathbf{a}_{4,k_4} \oplus \mathbf{a}_{3,k_3} = \tau \text{ and } \mathbf{a}_{2,k_2} \oplus \mathbf{a}_{1,k_1} \neq \mathbf{t} \oplus \tau \ .$$
Thus, when the for-loop (in line 1) with $\tau_1 := \mathsf{low}_\lambda(\tau)$, we store $\mathbf{a}_{4,k_4} \oplus \mathbf{a}_{3,k_3}$ in $L_{4,3}$ (see line 2). Meanwhile, $\mathbf{a}_{2,k_2} \oplus \mathbf{a}_{1,k_1}$ is not be picked in line 3 or line 4 since $\mathbf{a}_{2,k_2} \oplus \mathbf{a}_{1,k_1} \neq \mathbf{t} \oplus \tau$.

2. From the clause i to the clause $i+1$. $(k_{c_{i+1}}, \cdots, k_1) \in [2^\lambda]^{c_{i+1}}$ is a solution for target $\mathbf{t}$ iff there exists τ such that

$$\bigoplus_{j=c_i+1}^{c_{i+1}} \mathbf{a}_{j,k_j} = \tau \text{ and } \bigoplus_{j=1}^{c_i} \mathbf{a}_{j,k_j} = \mathbf{t} \oplus \tau \ .$$

If $\tau_{i+1} := \mathsf{low}_{(i+1)\cdot\lambda}(\tau)$ fixed in line 1, then we will store $\bigoplus_{j=c_i+1}^{c_{i+1}} \mathbf{a}_{j,k_j}$ in L_{c_{i+1},c_i+1} (see line 2). Meanwhile, $\bigoplus_{j=1}^{c_i} \mathbf{a}_{j,k_j}$ will be picked, because of $\mathsf{low}_{(i+1)\cdot\lambda}(\bigoplus_{j=1}^{c_i} \mathbf{a}_{j,k_j}) = \mathsf{low}_{(i+1)\cdot\lambda}(\mathbf{t}) \oplus \tau_{i+1}$.

□

Lemma 17. *For every $c_i \in \mathsf{magic}$ and $N = 2^\lambda$, in the c_i-Dissect (see Algorithm 8), it holds that*

$$\Pr\Big[|L(c_i, c_{i-1}+1)| \geq 2N\Big] \leq 1/N \ .$$

Proof. We prove it considering the two cases: $i > 0$ and $i = 0$, where the latter ($i = 0$) is the same as the case $i = 1$, and thus it suffices to prove the case $i > 0$.

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

$$\begin{aligned}\Pr\Big[X > 2N\Big] = \Pr\Big[X - \mathbb{E}[X] > N\Big] &\leq \Pr\Big[|X - \mathbb{E}[X]| \geq N\Big] \\ &\leq \frac{Var[X]}{N^2} = \frac{\sum_K Var[X_K]}{N^2} \leq \frac{\mathbb{E}[X]}{N^2} = \frac{1}{N} \ ,\end{aligned}$$

where the first equality is due to $N = 2^\lambda$ and $\mathbb{E}[X] = N^{i+1} \cdot 2^{-i\lambda} = N$, and the second inequality is based on Chebyshev's inequality, the second equality is due to Lemma 4, and the last inequality is due to

$$Var[X_i] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 \leq \mathbb{E}[X_i^2] = \mathbb{E}[X_i] \quad .$$

□

Lemma 18 (Space Consumption of the c_i-Dissect)**.** *For every $c_i \in \mathsf{magic}$, the c_i-Dissect (see Algorithm 8) requires space at most $N \cdot \mathsf{poly}(b, c_i)$ with the probability at least $1 - O(i)/N$.*

Proof. We only store the result list $L(c_i, c_{i-1}+1)$ from the join operation in line 2. Based on Lemma 17, we have $\Pr\big[|L(c_i, c_{i-1}+1)| \geq 2N\big] \leq 1/N$. Therefore, these lists can be computed using space of size at most $N \cdot \mathsf{poly}(b, c_i)$ (except with probability $1/N$), since the entries in $L_{c_i-1} \bowtie \cdots \bowtie L_{c_{i-1}+1}$ is computed on-the-fly and compared against L_{c_i}, using the adapted naive c-sum$^+$ algorithm.

In the base clause $i = 1$, the join of the lists L_1 and L_2 also can be computed in space $N \cdot \mathsf{poly}(b, c_i)$ with the probability at least $1 - 1/N$.

Furthermore, other intermediate elements passed from a recursive call (in line 3) are processed on-the-fly to the join operation (in line 4) and still on-the-fly passed to the c_{i+1}-Dissect (in line 6) without additional space. When pos = outer, the result will be dropped or added to $\mathcal{S}$ (see line 8) consuming space at most $N \cdot \mathsf{poly}(b, c_i)$ with the probability at least $1 - 2/N$ based on Lemma 5. In conclusion, the c_i-Dissect requires space of size up to $N \cdot \mathsf{poly}(b, c_i)$ with the probability at least $1 - O(i)/N$ by a union bound. □

Lemma 19 ([EHK+18]). *For every $c_i \in \mathsf{magic}$, one iteration of the outmost for-loop (see line 1) of c_i-Dissect$(\cdots, inner)$ (see Algorithm 8) will return at most $1.5 \cdot 2^{c_{i-2}\cdot\lambda}$ elements to the c_{i+1}-Dissect (in line 6) with the probability at least $1 - O(1)/N$.*

Lemma 19 is special case of Lemma 11 for $y = 0$, $\beta = 1$ with target length $(i+1)\cdot\lambda$, and thus can be proved following the steps of Lemma 11.

Lemma 20 (Running Time of the c_i-Dissect [EHK+18]). *For every $c_i \in \mathsf{magic}$, the c_i-Dissect (see Algorithm 8) runs in time at most $2^{c_{i-1}\cdot\lambda}\cdot\mathsf{poly}(b, c_i)$ with the probability at least $1-O(i)/N$.*

Proof. We prove it considering the two cases: pos = inner and pos = outer and the difference between the two cases is that the latter one should store all result elements, instead of passing on-the-fly to the c_{i+1}-Dissect (see lines 5-8). The store operation needs time at most $3N$ with the probability at least $1-2/N$, since the size of result elements can be bounded based on Lemma 5. Let $T^{outer}_{c_i}$ (resp. $T^{inner}_{c_i}$) denote the upper bound of running time of the c_i-Dissect$(\cdots, \text{outer})$ (resp. the c_i-Dissect$(\cdots, \text{inner})$) with the probability at least $1 - O(i)/N$ (resp. $1 - O(i)/N$). Thus, $T^{outer}_{c_i} \le T^{inner}_{c_i} + 3N$ via the union bound (see Lemma 1).

Here, we first analyse the running time of the c_i-Dissect$(\cdots, \text{inner})$. We prove it via an inductive approach.

1. The base clause $i = 1$. We observe that the for-loop (in line 1) iterates over 2^λ values. And in each iterate, the work includes 2 instances of adapted naive 2-sum$^+$ algorithm and binary search. Therefore, the time is $2^{2\cdot\lambda}\cdot\mathsf{poly}(b, c_i)$.

2. From the clause i to the clause $i+1$. For any $\tau_{i+1} \in \mathbb{F}_2^{(i+1)\cdot\lambda}$, each iteration of the for-loop (in line 1) over τ_{i+1} includes

   - line 2: an unbalanced join operation on $i+2$ lists $L_{c_{i+1}}, \cdots, L_{c_i+1}$ is computed in time $2^{(i+1)\cdot\lambda}\cdot\mathsf{poly}(b, c_i)$, since the entries in $L_{c_i-1} \bowtie \cdots \bowtie L_{c_{i-1}+1}$ is computed on-the-fly and compared against L_{c_i}, using the adapted naive c-sum$^+$ algorithm.
   - line 3: the c_i-Dissect$(\cdots, \text{inner})$ is called requiring time at most $T^{inner}_{c_i}$ with the probability at least $1 - i/N$
   - lines 4-6: at most $2\cdot 2^{c_{i-1}\cdot\lambda}$ partial distinct solutions are returned with the probability at least $1 - O(1)/N$, based on Lemma 19.

   Disregarding Oh-notation for convenience, we have,

$$\begin{aligned}\log T^{inner}_{c_{i+1}} &= \log\left(2^{(i+1)\cdot\lambda}\cdot\max\left\{2^{(i+1)\cdot\lambda}, T^{inner}_{c_i}, 2\cdot 2^{c_{i-1}\cdot\lambda}\right\}\right)\\ &= (i+1)\cdot\lambda + \max\{(i+1)\cdot\lambda, \log T^{inner}_{c_i}, c_{i-1}\cdot\lambda + 1\}\ .\end{aligned}$$

   Using the induction hypothesis we have $\log T^{inner}_{c_i} = c_{i-1}\cdot\lambda + 1$. Thus

$$\log T^{inner}_{c_{i+1}} = (i + 1 + \max\{(i+1)\cdot\lambda, c_{i-1}\cdot\lambda + 1\})\ .$$

   For $i \ge 1$, we have $i + 1 \le i + 1 + \frac{1}{2}(i^2 - i) = c_{i-1}$. Hence,

$$\begin{aligned}\log T^{inner}_{c_{i+1}} &= (i + 1 + c_{i-1})\cdot\lambda + 1\\ &= c_i\cdot\lambda + 1\ .\end{aligned}$$

Therefore, with probability at least $1-O(i)/N$, the running time of the c_i-Dissect$(\cdots, \text{inner})$ is $T^{inner}_{c_{i+1}} = 2^{c_{i-1}\cdot\lambda}\cdot\mathsf{poly}(b, c_i)$ via the union bound (see Lemma 1). Then, we analyze the running time of the c_i-Dissect$(\cdots, \text{outer})$. As discussed previously, $T^{outer}_{c_i} \le T^{inner}_{c_i} + 3N$. Therefore, with the the probability at least $1 - O(i)/N$, the running time of the c_i-Dissect$(\cdots, \text{outer})$ is $T^{outer}_{c_{i+1}} = 2^{c_{i-1}\cdot\lambda}\cdot\mathsf{poly}(b, c_i)$. □

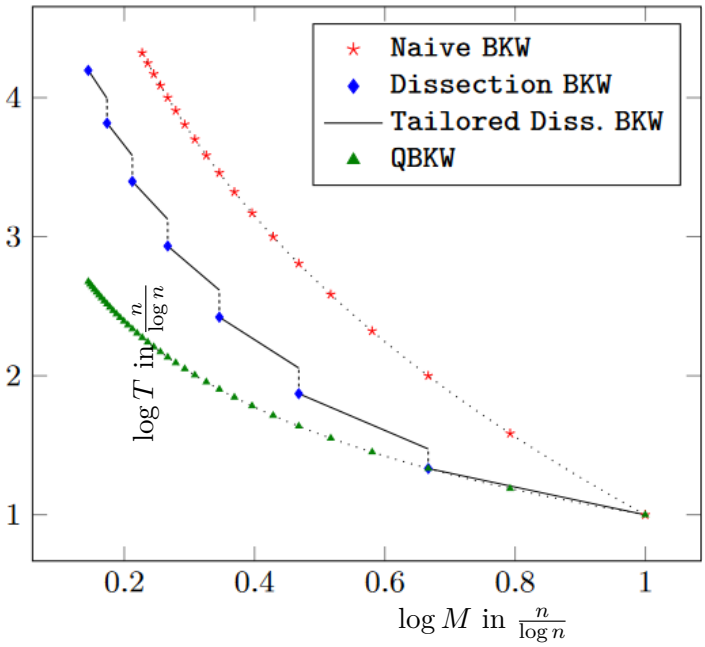


Figure 4: The time-space trade-offs for the variants of c-sum BKW ([EHK+18, Fig. 1]).

B The Tailored Dissection c-sum$^+$ BKW in Details

Tailoring Heuristic [EHK+18]. For every $c_i \in \mathsf{magic}$, let random variable Z_j denote the number of distinct solutions gained in the j-th iteration of the outmost for-loop of the c_i-Dissect (see line 1 in Algorithm 8) taken over the initial choice of input lists. Esser et al. [EHK+18] heuristically assume that there exists a polynomial function $\mathsf{poly}(\lambda)$, such that for all $\mathcal{J} \subset \{1, \cdots, 2^{i\lambda}\}$ we have

$$\Pr\left[\sum_{j\in\mathcal{J}} Z_j < \frac{1}{\mathsf{poly}(\lambda)} \cdot \mathbb{E}\Big[\sum_{j\in\mathcal{J}} Z_j\Big]\right] \leq \mathsf{negl}(\lambda) \ . \tag{3}$$

In particular, it follows from Equation 3 that for all $\iota \leq 2^{i\cdot\lambda}$ we have

$$\Pr\left[\sum_{j=1}^{\iota\cdot\mathsf{poly}(\lambda)} Z_j \geq \mathbb{E}\Big[\sum_{j=1}^{\iota} Z_j\Big]\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

Proof of Lemma 11. Assume $\tau = \{\tau_{i,1}, \cdots, \tau_{i,2^y}\}$ is the set of the constraints for the first 2^y-th iterations of the outmost loop (see line 1) of the c_i-Dissect (see Algorithm 8). For every $K = (k_{c_{i-1}+1}, \cdots, k_{c_i}) \in [N^\beta]^{i+1}$ define an indicator variable X_K that takes value $X_K = 1$ iff $\mathsf{low}_{i\cdot\lambda}(\bigoplus_{j=c_{i-1}+1}^{c_i} \mathbf{a}_{j,k_j}) \in \tau$. Thus, $X = \sum_K X_K$ is the sum of the size of the intermediate list $L_{(c_i,c_{i-1}+1)}$ over the constraint set τ, where every $K \in [2^\lambda]^{i+1}$ has expectation $\mathbb{E}[X_K] = 2^{y-i\cdot\lambda}$ and all the X_K are pairwise independent. Therefore, similar with the proof of Lemma 5, we have

$$\Pr\left[\left||X| - 2^{y+\lambda}\right| \leq \frac{1}{4}\cdot 2^{y+\lambda}\right] \geq 1 - 16\cdot 2^{-(y+\lambda)}$$

Let set $\mathcal{K} \stackrel{\mathsf{def}}{=} \{K | K \in [2^\lambda]^{i+1} \text{ and } X_K = 1\}$. For every $L = (l_1, \cdots l_{c_i}) \in [2^\lambda]^{c_{i-1}} \times \mathcal{K}$ define an indicator variable that takes value $Z_L = 1$ iff $\bigoplus_{j=1}^{c_i} \mathbf{a}_{j,l_j} = \mathbf{t}$, thus, for every $L \in [2^\lambda]^{c_{i-1}} \times \mathcal{K}$, we have $\mathbb{E}[Z_L] = 2^{-b}$ and all the Z_L are pairwise independent. Let $Z = \sum_L Z_L$ be the number of distinct solutions obtained in the first 2^y-th iterations of the outmost loop of the c_i-Dissect.

Hence, $\mathbb{E}[Z] = 2^{c_{i-1}\cdot\lambda} \cdot |\mathcal{K}| \cdot 2^{-b}$. Therefore, we have

$$\begin{aligned}
&\Pr\left[\left|\mathbb{E}[Z] - 2N^\beta\right| \le N^\beta/2\right] \\
=&\Pr\left[\left|\mathbb{E}[Z] - 2^{y+\lambda}\cdot 2^{c_{i-1}\cdot\lambda}\cdot 2^{-b}\right| \le \frac{1}{4}\cdot 2^{y+\lambda}\cdot 2^{c_{i-1}\cdot\lambda}\cdot 2^{-b}\right] \\
=&\Pr\left[\left||X| - 2^{y+\lambda}\right| \le \frac{1}{4}\cdot 2^{y+\lambda}\right] \ge 1 - 16\cdot 2^{-(y+\lambda)} \ge 1 - 16/N^\beta \ ,
\end{aligned}$$

where the first equality is due to $y := b - c_{i-1}\cdot\lambda + 1$ and $2^{y+\lambda}\cdot 2^{c_{i-1}\cdot\lambda}\cdot 2^{-b} = 2^{b-c_{i-1}\cdot\lambda+1+\lambda+c_{i-1}\cdot\lambda-b} = 2^{\lambda+1} = 2N^\beta$ and first inequality is due to $y \ge 0$.

$$\begin{aligned}
&\Pr\left[|Z - 2N^\beta| > N^\beta\right] \le \Pr\left[|Z - \mathbb{E}[Z]| \ge N^\beta/2\right] + \Pr\left[\left|\mathbb{E}[Z] - 2N^\beta\right| > N^\beta/2\right] \\
\le&\frac{Var[Z]}{N^{2\beta}/4} + 16/N^\beta = \frac{\sum_L Var[Z_L]}{N^{2\beta}/4} + 16/N^\beta \le \frac{\sum_L \mathbb{E}[Z_L]}{N^{2\beta}/4} + 16/N^\beta \le 42/N^\beta
\end{aligned}$$

where the second inequality is by Chebyshev's inequality, and the first equality is due to that for pairwise independent r.v.s $Z_1, \cdots, Z_m$

$$Var[\sum_{i=1}^{m} Z_i] = \sum_{i=1}^{m} Var[Z_i] \ ,$$

the third inequality is due to

$$Var[Z_i] = \mathbb{E}[Z_i^2] - \mathbb{E}[Z_i]^2 \le \mathbb{E}[Z_i]$$

and the fourth inequality is due to $\Pr\left[\left|\mathbb{E}[Z] - 2N^\beta\right| \le N^\beta/2\right] \ge 1 - 16/N^\beta$. □

Proof of Lemma 12. Let $\lambda = \frac{\beta\cdot(b+1)}{c_i - 1}$ as defined in Algorithm 8. Since the tailored Dissection c_i-sum$^+$ algorithm (see Algorithm 5) only invokes the c_i-Dissect, then the analysis of the space and time complexities is similar with that of the c_i-Dissect. Hence, similar with the proof of Lemma 18, we have

$$\Pr[M \le N^\beta \cdot \mathsf{poly}(b, c_i)] \ge 1 - O(i)/N^\beta \ ,$$

since the size of intermediate list and result elements can be bounded by Lemma 17 and Lemma 11.

Let $T_{c_i}^{tailored}$ (resp. $T_{c_i}^{inner}$) denote the upper bound of running time of Algorithm 5 (resp. the c_i-Dissect$(\cdots, \text{inner})$) with the probability at least $1 - O(i)/N^\beta$ (resp. $1 - O(i)/N^\beta$). Based on the analysis of running time of the c_i-Dissect (see Lemma 20), we can obtain that

$$\log T_{c_i}^{inner} = c_{i-1}\cdot\lambda + 1 \ .$$

Based on Lemma 11, Algorithm 5 halts in the first 2^y-th iterations of the outmost loop (see line 1) of the c_i-Dissect (see Algorithm 8) and at most $3N^\beta$ distinct solutions are returned with the probability at least $1 - O(1)/N^\beta$. Therefore,

$$\log T_{c_i}^{tailored} = \max\{y + \log T_{c_{i-1}}^{inner}, \log(3N^\beta)\} \ .$$

Based on $N^\beta = 2^\lambda$, $\log T_{c_{i-1}}^{inner} = c_{i-2}\cdot\lambda + 1$ and $y := b - c_{i-1}\cdot\lambda + 1$, we have

$$\begin{aligned}
\log T_{c_i}^{tailored} &= \max\{b - c_{i-1}\cdot\lambda + c_{i-2}\cdot\lambda + 2, \lambda + \log 3\} \\
&= b - i\cdot\lambda + 2
\end{aligned}$$

Further, because of $\lambda = \frac{\beta\cdot(b+1)}{c_i - 1}$ (i.e., $b + 1 = \frac{(c_i-1)\cdot\lambda}{\beta}$), then

$$\log T_{c_i}^{tailored} = \frac{(c_i - 1)\lambda}{\beta} - i\cdot\lambda + 1 \ .$$

Based on $\lambda = \beta \cdot \log N$, we have $T_{c_i}^{tailored} = N^{c_{i-1}-(\beta-1)\cdot i} \cdot \mathsf{poly}(b, c_i)$.

Therefore, the tailored Dissection c_i-sum$^+$ algorithm finds N^β distinct solutions in time $T = N^{c_{i-1}-(\beta-1)\cdot i} \cdot \mathsf{poly}(b, c_i)$ and space $M = N^\beta \cdot \mathsf{poly}(b, c_i)$ with the probability at least $1 - O(i)/N$ via the union bound (see Lemma 1).

□

C Algorithms, Lemmas and Proofs

Lemma 21 (Chernoff Bound)**.** *Let $X_1, \cdots, X_n$ be independent variables with $0 \le X_i \le 1$ for all $1 \le i \le n$, denote $\mu = \mathbb{E}[(\sum_{i=1}^n X_i)/n]$. Then, for any $\epsilon > 0$*

$$\Pr\left[\left|\frac{\sum_{i=1}^n X_i}{n} - \mu\right| > \epsilon\right] < 2^{-\epsilon^2 \cdot n} \quad .$$

Proof of Lemma 4. We have

$$Var\Big[\sum_{i=1}^m X_i\Big] = \mathbb{E}\Big[(\sum_{i=1}^m X_i)^2\Big] - \mathbb{E}\Big[\sum_{i=1}^m X_i\Big]^2 = \sum_{i=1}^m (\mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2) = \sum_{i=1}^m Var[X_i]$$

where we recall $\mathbb{E}[X_i \cdot X_j] = \mathbb{E}[X_i] \cdot \mathbb{E}[X_j]$ for independent X_i and X_j. □

Lemma 22. *For $0 < d < 1/c$ and $2 \le c \in \mathbb{N}$ we have $(1+d)^c \le 1 + 2cd$, and $(1-d)^c \ge 1 - cd$.*

Proof. The second follows from Bernoulli's inequality, and the first is due to

$$\begin{aligned}
(1+d)^c &= 1 + cd(1 + \frac{(c-1)}{2!}d + \frac{(c-1)(c-2)}{3!}d^2 + \cdots + \frac{(c-1)(c-2)\cdots 1}{c!}d^{c-1}) \\
&\le 1 + cd(1 + \frac{1}{2!} + \cdots + \frac{1}{c!}) \\
&\le 1 + cd(1 + \frac{1}{2 \times 1} + \cdots + \frac{1}{c(c-1)}) \\
&= 1 + cd(2 - \frac{1}{c}) \le 1 + 2cd \ .
\end{aligned}$$

□

Algorithm 9: The exhaustive c-sum$^+$

Input: $L_c, \cdots, L_1 \in (\mathbb{F}_2^b)^N$ and $\mathbf{t} \in \mathbb{F}_2^b$, where $L_j \stackrel{\mathsf{def}}{=} (\mathbf{a}_{j,1}, \cdots, \mathbf{a}_{j,N})$ for $j \in [c]$
Output: $\mathcal{S} \subset \binom{[N]}{c}$

```
Sort out L_c ;
for all V = (i_1, ..., i_{c-1}) ∈ [N]^{c-1} do
    s = ⊕_{j∈[c]} a_{j,i_j};
    for all i_c ∈ [N] satisfying a_{c,i_c} = t ⊕ s do
        S ← S ∪ {{i_1, ..., i_c}};
Return S;
```