On Iterative Collision Search for LPN and Subset Sum

Srinivas Devadas Ling Ren Hanshen Xiao

Massachusetts Institute of Technology, Cambridge, MA
{devadas, renling, hsxiao}@mit.edu

Abstract. Iterative collision search procedures play a key role in developing combinatorial algorithms for the subset sum and learning parity with noise (LPN) problems. In both scenarios, the single-list pair-wise iterative collision search finds the most solutions and offers the best efficiency. However, due to its complex probabilistic structure, no rigorous analysis for it appears to be available to the best of our knowledge. As a result, theoretical works often resort to overly constrained and sub-optimal iterative collision search variants in exchange for analytic simplicity. In this paper, we present rigorous analysis for the single-list pair-wise iterative collision search method and its applications in subset sum and LPN. In the subset sum direction, it outperforms Wagner's algorithm for attacking knapsack-based cryptosystems. In the LPN literature, the single-list pair-wise iterative collision search method is known as the LF2 heuristic. Besides LF2, we also present rigorous analysis of other LPN solving heuristics and show that they work well when combined with LF2. Putting it together, we significantly narrow the gap between theoretical and heuristic algorithms for LPN.

1 Introduction

The Learning Parity with Noise (LPN) problem is a fundamental problem in coding theory, cryptography and machine learning. In cryptography, LPN attracts most interest from lightweight constructions, i.e., those that run efficiently on constrained devices such as RFID tags and wireless sensors. Many lightweight constructions [11,13,9,15] build on the hardness of the LPN due to the simplicity of the operations it entails. Studying the best algorithms for solving LPN is vital to determine suitable parameters for these constructions and subsequent improvements.

For a uniformly selected secret $\mathbf{s} \in \mathbb{Z}_2^n$, the LPN problem is to find $\mathbf{s}$ given input samples $\mathbf{As}+\mathbf{e}$, where $\mathbf{A}$ is uniformly random and each component of $\mathbf{e}$ is a Bernoulli noise. For ease of exposition, we follow prior work and think of LPN algorithms as consisting of two phases: a reduction phase and a solving phase. The classical algorithm for LPN is the BKW algorithm [6]. At its core is an iterative collision search procedure for the reduction phase. To start, partition the samples into $2^{\frac{n}{k+1}}$ groups such that the first $\frac{n}{k+1}$ bits are identical. Here, k is a parameter of the algorithm and is set to $\Theta(\log n)$. Then, select one sample in each group and add it to the other ones in the group to cancel out the first $\frac{n}{k+1}$ bits. Each subsequent iterative step follows the same procedure to cancel out the next $\frac{n}{k+1}$ bits. After a few iterations, the samples only depend on a single bit in the secret. These samples are the outputs of the reduction phase and we call them *reduced* samples. At this point, the algorithm enters the solving phase to guess this secret bit and tests it on the reduced samples. The algorithm then moves on to guess the next secret bit, repeating the reduction phase and the solving phase therein.

The BKW algorithm needs a sub-exponential number of input samples. Lyubashevsky [19] and Kirchner [16] modified the BKW algorithm to work with a polynomial number of samples. Outside the "limited-sample" direction, however, theoretical advances for LPN algorithms have been stagnant for more than a decade. On the other hand, heuristic and practical methods for

LPN continue to develop at a fast pace. Levieil and Fouque [17] proposed two important heuristic methods. The first one, LF1, improves the solving phase by guessing multiple secret bits at a time. It is augmented with the Fast Walsh-Hadamard transform to further reduce runtime. The second method, LF2, is a more efficient iterative collision search procedure in the reduction phase. The goal is to generate more reduced samples for the solving phase. After partitioning input samples into groups sharing a chunk of bits, instead of adding one sample to the others in the group as in BKW, LF2 computes the sums of every pair in the group. Recent works [17,10,27,7,8] have applied covering codes, partial secret guessing and linear programming to improve the solving phase.

The LF1 and LF2 heuristics are two most important heuristic techniques in the LPN literature, and have been adopted by every subsequent work we know of [10,27,7,8]. The efficiency gain, however, presents a challenge for analysis since the reduced samples now depend on each other in a complex manner. (LF1 was initially presented as a rigorous algorithm [17], but Zhang et al. [27] pointed out that the original proof incorrectly assumed independence between reduced samples. Hence, LF1 should be treated as a heuristic prior to our work.) A main contribution of this paper is to provide rigorous analysis for the LF1 and LF2 methods and establish them as rigorous LPN algorithms. In particular, we compute the number of solutions (both expectation and distribution) produced by LF2 in the reduction phase. We also show that the correlation between LF2 reduced samples has little impact on the success rate of the LPN solving phase for both majority voting and LF1 Walsh-Hadamard transform. Our results significantly narrow the gap between theoretical and heuristic solutions to the LPN problem.

LPN has a close connection to the subset sum problem. As Wagner suggests [26], any improvement to the subset sum problem will also result in an improvement to LPN. In this paper, we consider the random fixed-weighted XOR variant of subset sum. Given a list L of elements sampled uniformly randomly from $\mathbb{Z}_2^n$, find 2^k elements from L such that they XOR to 0. Most works [21,2,4] treat Wagner's algorithm [26] as the default and classical algorithm for this problem. However, Wagner's algorithm was not tailored for fixed weighted subset sum. Instead, it was presented for the generalized birthday problem [26]. In the generalized birthday problem, there are 2^k separate lists and the goal is to find one element from each list such that they XOR to 0. In order to apply Wagner's algorithm, prior works have to partition the single list L into 2^k smaller lists. Wagner's algorithm then places the 2^k lists as the leaves of a depth-k binary tree. In step i, every pair of sibling lists are merged into a new list at their parent node such that the i-th chunk of $\frac{n}{k+1}$ bits are canceled out. To elaborate, the merge operation searches for two elements, one from each input list, such that their i-th chunk of $\frac{n}{k+1}$ bits XOR to 0. After k steps, the elements in the last list at the root of the tree are solutions to the problem.

Clearly, the partition into 2^k separate lists is an artifact in order to invoke Wagner's algorithm. It not only increases the time complexity but also imposes an unnecessary constraint that eliminates many valid candidate solutions. It is much more natural to perform the same merge operation within the original single list L: at step i, search for pairs of distinct elements in L that cancel out the i-th chunk of $\frac{n}{k+1}$ bits, and add their XOR results to the new list for the next step. This single-list pair-wise iterative collision search very much resembles the LF2 method (there are also important differences which we describe in Section 3.2). Also resembling LF2, it creates difficulties for the analysis. In Wagner's algorithm, in every merge operation, the two input elements (from different lists) are independent of each other. In contrast, the single-list iterative collision search introduces dependence across steps, making it hard to reason about the expected list size after each step or the number of solutions produced in the end. With a rigorous analysis, we establish

the single-list pair-wise iterative collision search as an improved algorithm over Wagner for random fixed weighted subset sum. Cryptographic constructions based on it need to adjust their parameter choices accordingly [21,2].

The rest of the paper is organized as follows. We start with the fixed weighted subset sum problem since the LPN problem additionally has to deal with the solving phase. Section 2 presents our analysis for the single-list iterative collision search algorithm for the fixed weighted subset sum problem. Section 3 presents our analysis for the LF1 and LF2 methods for LPN. We conclude in Section 4.

2 Random Fixed Weighted Subset Sum

2.1 Background

Definition 1 (subset sum). *Given a list $L = \{a_1, a_2, ..., a_N\}$ of N numbers from an algebraic structure and an operation $\oplus$, find $\mathbf{x} \in \{0,1\}^N$ such that $\langle \mathbf{x}, S \rangle = x_1 a_1 \oplus x_2 a_2 \oplus ... \oplus x_N a_N = t$ where t is a pre-defined target.*

The subset sum problem is one of Karp's 21 NP-complete problems [14]. The classical subset sum problem considers integers and integer addition. In the last three decades, there have also been a few important variants of the subset sum problem that attracted interest in cryptography [18,20,12].

In this paper, we focus on the random fixed weighted variant of the problem. For concreteness, we start with the XOR case and consider a larger modulus q later. Specifically, $a_1, a_2, \cdots, a_N$ are n-bit binary string drawn independently and uniformly randomly from $\mathbb{Z}_2^n$. The operator $\oplus$ is bit-wise XOR. The solution vector $\mathbf{x}$ must have a Hamming weight of 2^k. We also focus on the special case where the target is $t = 0$.

Wagner's Generalized Birthday Problem and Algorithm. Wagner introduced the generalized birthday problem and an algorithm for it [26]. The generalized birthday problem bears some similarities to the random fixed-weighted subset sum problem, but is also different in a fundamental way. Instead of finding 2^k elements from a single list, the problem takes 2^k lists and finds one element from each list.

Definition 2 (generalized birthday problem). *Given 2^k lists $L_1, L_2, \cdots, L_{2^k}$ each containing N elements in $\mathbb{Z}_2^n$, find one element from each list $a_1 \in L_1, a_2 \in L_2, \cdots, a_{2^k} \in L_{2^k}$ such that $a_1 \oplus a_2 \oplus \cdots \oplus a_{2^k} = 0$.*

Wagner's algorithm performs iterative collision search in a tree fashion in k steps. [1] Write the 2^k input lists as $L_1^{(0)}, L_2^{(0)}, ..., L_{2^k}^{(0)}$ and place them at the leaves of a binary tree of depth k. In the j-th step ($1 \leq j < k$), for each pair of lists $L_{2i}^{(j-1)}$ and $L_{2i+1}^{(j-1)}$, find two elements $l \in L_{2i}^{(j-1)}$ and $l' \in L_{2i+1}^{(j-1)}$ such that the j-th chunk of $\frac{n}{k+1}$ bits cancel out (i.e., XOR to 0), and then add $l \oplus l'$ to a new list $L_i^{(j)}$. In the last step $j = k$, there are only two lists remaining, and the algorithm looks for two elements, one from each list, such that they cancel out the last $\frac{2n}{k+1}$ bits and XOR to 0^n. Figure 1 gives an illustration of this algorithm. There have been several improvements to Wagner's algorithm [3,22,16], and they all follow the tree-based collision search framework.

[1] Different from our notation, Wagner denoted the number of lists as k and the number of steps as $\log_2 k$ [26].

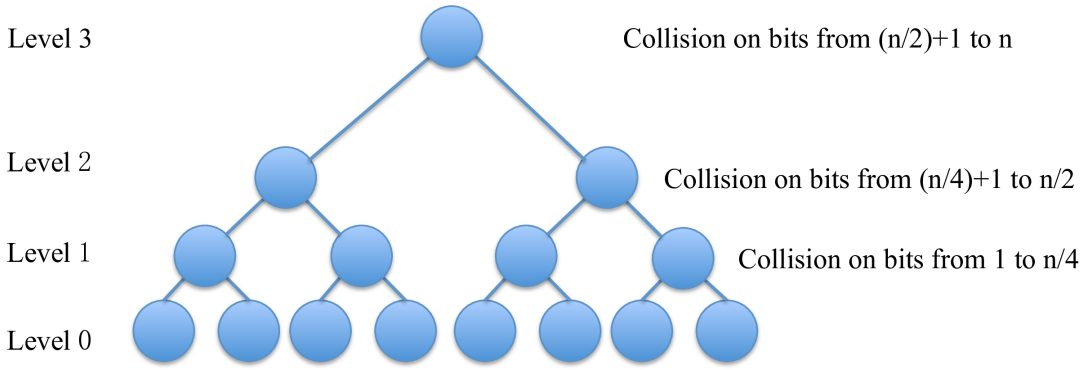

Fig. 1: An illustration of Wagner's algorithm.

Input: A single list L, also written as $L^{(0)}$, of size N.
1. Initially, add the index alongside each element in $L^{(0)}$, i.e., each element in $L^{(0)}$ now has the form $(a_i, \{i\})$.
2. **for** $j = 1 : k-1$ **do**
   For each pair of elements (a, α) and (a', α') in $L^{(j-1)}$, if $a \oplus a'$ cancel out the j-th chunk of $\frac{n}{k+1}$ bits and $\alpha \cap \alpha' = \varnothing$, then add $(a \oplus a', \alpha \cup \alpha')$ to the new list $L^{(j)}$.
end
3. At the last step, repeat the similar operation to find a pair of elements (a, α) and (a', α') such that $a \oplus a'$ cancel out the last $\frac{2n}{k+1}$ bits and $\alpha \cap \alpha' = \varnothing$. Output $L^{(k)}$.

Algorithm 1: The single-list pair-wise iterative collision search algorithm.

To ensure at least one solution is found in expectation, the size of each input list should be at least $N \geq 2^{\frac{n}{k+1}}$. Crucially for the analysis, in each step, a pair of elements l and l' are independent because they are sums of elements that come from disjoint lists. Thus, the expected list size at each step can be easily calculated as $N^2 \cdot 2^{-\frac{n}{k+1}} \geq N$, and in the last step, $N^2 \cdot 2^{-\frac{2n}{k+1}} \geq 1$ solutions are produced in expectation.

2.2 Single List Iterative Collision Search

While Wagner's algorithm is clearly tailored for the generalized birthday problem, many previous works [21,2,4,16] apply it to the random fixed-weighted subset sum problem. To do so, they have to artificially divide the list L into 2^k disjoint lists $L_1^{(0)}, L_2^{(0)}, ..., L_{2^k}^{(0)}$. As mentioned, the division imposes the unnecessary restriction of finding one element from each list. It not only eliminates many valid solutions but also increases space and time complexity.

It is much more natural to perform the iterative collision search directly on a single list without division. In particular, we repeatedly merge the single list L with itself, i.e., find pairs of elements in L that cancel out the next chunk of bits, and add their XOR results to a new list for the next step. Algorithm 1 gives the pseudocode.

The single-list pair-wise collision search algorithm is known as the LF2 method in the LPN literature [17], which was in turn inspired by Wagner's algorithm [26]. However, it has not gained much attention in knapsack-based cryptosystems and cryptanalysis where it is most suitable. Recently, it was also independently proposed, though seemingly by accident, in a memory hard proof-of-work scheme called Equihash [5]. The Equihash paper [5] used the above Algorithm 1 to solve the random

fixed-weighted subset sum, but confusingly, claimed to be using Wagner's algorithm and solving the generalized birthday problem throughout the paper.

As we have mentioned, analyzing the single-list pair-wise collision search algorithm is much harder than analyzing Wagner's algorithm because, after the first step, elements in the list become correlated. They are no longer sums of non-overlapping elements. Rather, they are now sums that contain common addends. If the input list size at a certain step is N, the expected output list size is no longer simply $N^2 \cdot 2^{-\frac{n}{k+1}}$. Indeed, it seems difficult to derive the final expected number of solutions by calculating the expected list size at each step. In the next subsection, we approach the problem from a different angle. We will calculate the total number of distinct candidate solutions and the probability that each one is an actual solution that Algorithm 1 produces.

2.3 Expected Number of Solutions

Theorem 1. *Let $p = 2^{-\frac{n}{k+1}}$. The expected number of solutions produced by Algorithm 1 is*

$$\mathrm{E}\left[|L^{(k)}|\right] = 2\binom{N}{2^k}(2^k)!(p/2)^{2^k}.$$

Proof. Consider an index vector $\alpha = (i_1, i_2, ..., i_{2^k})$, and the candidate solution it defines, $\mathbf{a}_\alpha = \{a_{i_1}, a_{i_2}, \cdots, a_{i_{2^k}}\}$. Let $Y_\alpha = 1$ if $\mathbf{a}_\alpha$ is a solution produced by Algorithm 1 and $Y_\alpha = 0$ otherwise. Before we proceed, we remark that a solution to the fixed-weighted subset sum problem is not necessarily a solution that will be found by Algorithm 1. (The other direction is true). The reason is that Algorithm 1 can only find solutions that meet stringent conditions, i.e., those that cancel out a chunk of bits after each step. For example, if $\alpha = (1, 2, 3, 4)$ and $Y_\alpha = 1$, it is not only required that $a_1 \oplus a_2 \oplus a_3 \oplus a_4 = 0$, but also that $a_1 \oplus a_2$ and $a_3 \oplus a_4$ both cancel out the first chunk of bits. The iterative collision search framework in general only finds solutions with a specific structure rather than all solutions.

It is also important to note that some index vectors represent the same solution and should be counted only once. For example, if $\alpha = (1, 2, 3, 4)$ and $Y_\alpha = 1$, then for $\alpha' = (1, 2, 4, 3)$, $(2, 1, 3, 4)$, $(2, 1, 4, 3)$, $(3, 4, 1, 2)$, $(3, 4, 2, 1)$, $(4, 3, 1, 2)$, or $(4, 3, 2, 1)$, we have $Y_{\alpha'} = 1$. However, these eight vectors all represent the same single solution that will be produced by Algorithm 1. Define I to be a maximal set of index vectors that correspond to distinct candidate solutions. To calculate $|I|$, we think of the indices in a vector as the leaves in a binary tree of depth k. (This binary tree is just a tool for analyzing Algorithm 1 and should not be confused with Wagner's tree-based iterative collision search in Figure 1.) In the j^{th} step, swapping the two siblings would yield the same candidate solution. Thus, for each subset of 2^k elements, there are $\frac{(2^k)!}{\prod_{j=1}^{k} 2^{k-i}} = \frac{(2^k)!}{2^{2^k-1}}$ distinct candidate solutions out of the $(2^k)!$ total possible index vectors. Therefore, $|I| = \binom{N}{2^k}\frac{(2^k)!}{2^{2^k-1}}$ is the number of distinct candidate solutions that Algorithm 1 can possibly produce. The expected number of solutions produced by Algorithm 1 can then be calculated as $\mathrm{E}\left[|L^{(k)}|\right] = \mathrm{E}\left[\sum_{\alpha \in I} Y_\alpha\right]$.

After the j^{th} step, the list $L^{(j)}$ contains (XOR) sums of 2^j addends. We again think of the 2^j addends as leaves of a binary tree of depth j. To appear in $L^{(j)}$, the two addends need to cancel out a chunk of $\frac{n}{k+1}$ bits at each node in the tree. At each node, the probability is[2] $p = 2^{-\frac{n}{k+1}}$ and

[2] $p = 2^{-\frac{n}{k+1}}$ is used throughout the paper.

Table 1: The expected number of solutions found through experiments and Theorem 1.

n	16	32	48	56	96	128	160	192
k	1	3	5	6	5	7	9	11
Experiments	2.00	1.90	0.76	0.03	2.00	1.8	0.8	0.0
Theorem 1	1.9961	1.8931	0.7437	0.0328	1.9924	1.8797	0.7362	2.1×10^{-7}

there are $2^j - 1$ nodes in a tree of depth j. So the probability that a sum of certain 2^j addends appear in $L^{(j)}$ is p^{2^j-1}. The expected number of elements in $L^{(j)}$ is hence

$$\mathrm{E}\left[|L^{(j)}|\right] = |I| \cdot p^{2^j-1} = \binom{N}{2^j} \frac{(2^j)!}{2^{2^j-1}} \cdot p^{2^j-1} = \binom{N}{2^j} (2^j)!(p/2)^{2^j-1}. \tag{1}$$

The last step needs to cancel out $\frac{2n}{k+1}$ bits which happens with probability p^2. Thus, we have $\Pr(Y_\alpha = 1) = p^{2^k}$, and

$$\mathrm{E}\left[|L^{(k)}|\right] = |I| \cdot p^{2^k} = \binom{N}{2^k} \frac{(2^k)!}{2^{2^k-1}} \cdot p^{2^k} = 2\binom{N}{2^k} (2^k)!(p/2)^{2^k}$$

Remark. Although we presented our analysis in the XOR case for simplicity, it can be easily modified to work with a larger modulus q, i.e., when the operator $\oplus$ is modular addition over $\mathbb{Z}_q^n$.

2.4 Experimental Verification

In this subsection, we provide experimental results that corroborate the expected number of solutions we derive in Theorem 1. Another purpose of this section is to correct a mistake in the Equihash scheme [5]. Specifically, Equihash adopts Algorithm 1 with a list size $N = 2^{\frac{n}{k+1}+1}$. It then claimed the expected number of solutions is $\binom{N}{2} \cdot 2^{-\frac{2n}{k+1}} \approx 2$ citing Wagner's analysis. As we mentioned, Wagner's analysis requires independence and does not hold in the single-list case.

Table 1 lists the expected number of solutions found through experiments as well as the values given by Theorem 1 under different choices of n and k. Our theorem accurately predicts the number of solutions. (Our theorem is precise. The difference is due to errors in the experiments.) Equihash claims 2 solutions in expectation under all parameter settings, which as we see can be orders of magnitude off. We note that the latter four (n, k) pairs are among the recommended parameter settings from the Equihash paper [5]. For readers who are interested, this incorrect estimation will make the difficulty of the proof-of-work scheme proportionally harder than intended. For example, if a protocol designer adopts Equihash with $(n, k) = (192, 11)$, the expected time to find a valid proof-of-work will be $10^7\times$ longer than intended!

2.5 Distribution of Solutions

Knowing the expected number of solutions is in most cases sufficient to parameterize an algorithm. For example, to attack knapsack-based cryptosystems, one may parameterize Algorithm 1 to produce a small constant number of solutions in expectation, e.g., 1. But for a rigorous analysis, we would like to rule out a possible bad corner case. With the expectation being 1, it is possible that Algorithm 1 generates 2^{30} solutions with a 2^{-30} probability, while producing no solution most of

the time. In this subsection, we study the distribution of the number of solutions produced by Algorithm 1. Aside from ruling out that bad corner case, a more precise distribution will be useful in our analysis for LPN and possibly other applications.

We will show that the distribution of solutions is close to a Poisson distribution. We will apply the Chen-Stein method of the second moment analysis as the main tool to bound the difference.

Lemma 1 (Chen-Stein [1]). *Let Π be a random variable that follows a Poisson distribution with mean $\lambda = \mathrm{E}\left[|L^{(k)}|\right]$. Let J_α be the neighborhood of dependence for Y_α (which means any $Y_\beta \notin J_\alpha$ is independent of Y_α) and $J^*_\alpha = J_\alpha \setminus \{\alpha\}$ where $\setminus$ is set subtraction. Then,*

$$\begin{aligned}&\sum_{j=0}^{\infty}\left|P\left(|L^{(k)}| = j\right) - P(\Pi = j)\right| \\ \leq &\frac{4(1-e^{-\lambda})}{\lambda}\left(\sum_{\alpha\in I}\sum_{\beta\in J_\alpha}\mathrm{E}\left[Y_\alpha\right]E(Y_\beta] + \sum_{\alpha\in I}\sum_{\beta\in J^*_\alpha}\mathrm{E}\left[Y_\alpha Y_\beta\right]\right).\end{aligned}$$

The rest of this subsection bounds the two double sums separately. The first sum is

$$\begin{aligned}&\sum_{\alpha\in I}\sum_{\beta\in J_\alpha}\mathrm{E}\left[Y_\alpha\right]\mathrm{E}\left[Y_\beta\right] = \mathrm{E}\left[|L^{(k)}|\right]\sum_{\beta\in J_\alpha}E(Y_\beta) \\ &= \mathrm{E}\left[|L^{(k)}|\right]p^{2^k}\frac{(2^k)!}{2^{2^k-1}}\sum_{i=0}^{2^k-1}\binom{2^k}{i}\binom{N-2^k}{i} \approx \mathrm{E}\left[|L^{(k)}|\right]\cdot p.\end{aligned}$$

In most applications (e.g., attack hash functions), finding a few solutions is sufficient, so $\mathrm{E}\left[|L^{(k)}|\right]$ will be much less than $\frac{1}{p}$, and this first sum can be ignored.

The dominant part and also the difficulty of this analysis is the sum of the correlation terms $E(Y_\alpha Y_\beta)$. To start, we have

$$\begin{aligned}\mathrm{E}\left[Y_\alpha Y_\beta\right] &= \mathrm{E}\left[\mathrm{E}\left[Y_\alpha Y_\beta | Y_\beta\right]\right] = \Pr(Y_\alpha = 1, Y_\beta = 1) \\ &= \Pr(Y_\alpha = 1)\Pr(Y_\beta = 1 | Y_\alpha = 1).\end{aligned}$$

The last term above depends on the overlap pattern between two index vectors (and their corresponding candidate solutions). For convenience, we denote a candidate solution by an index vector, e.g., $\alpha = (1,2,3,4)$ refers to the candidate solution $\{a_1, a_2, a_3, a_4\}$. We again treat their elements as leaves of a binary tree. For each node in the tree for β, we color it black if its XOR output is independent of α. At the leaf level, any element in β that does not appear in α is independent of α and is colored black. For each level above, a node is colored black if at least one of its two children is black. This is because XORing with an independent and uniformly random addend yields an independent and uniformly random output.

As black nodes in β's tree are independent of α, we have $P(Y_\beta = 1 | Y_\alpha = 1) \leq p^{1+B}$ where B is the number of black nodes in the tree excluding the leaf level. This is because the candidate solution β needs to cancel out a chunk of bits at each node, and what happens with β at the black nodes are independent of α. The extra p is because the last step (the tree root) cancels out $\frac{2n}{k+1}$ bits which happens with p^2. p^{1+B} reaches its largest value when there are fewest black nodes in the tree. For

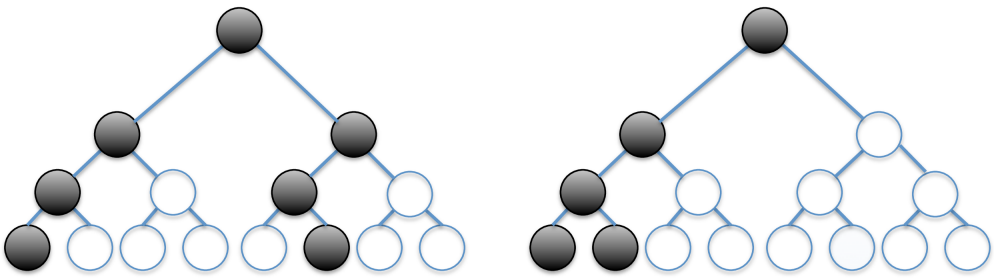

Fig. 2: Fewest black nodes occur when black nodes at the leaf level are clustered in the smallest subtree possible.

a certain number of black nodes at the leaves (height 0), the number of black nodes in the entire tree is the fewest if all the black nodes at height 0 are contained in the smallest subtree possible. Figure 2 gives an illustration of this configuration with the minimal number of black nodes. In this case, an upper bound on the number of black nodes in the tree can be derived as follows:

$$\begin{cases} \gamma_0 = |\beta \setminus \alpha| & \\ \gamma_j = \lceil \gamma_{j-1}/2 \rceil & j > 0 \\ \gamma(m) = \sum_{j=1}^{k} \gamma_j \quad \text{where} & m = \gamma_0 \end{cases} \tag{2}$$

Then, $\gamma\left(|\beta \setminus \alpha|\right)$ is an upper bound on B, and we have

$$\mathrm{E}\left[Y_\alpha Y_\beta\right] \leq \Pr(Y_\alpha = 1) \Pr(Y_\beta = 1 | Y_\alpha = 1) \leq p^{2^k+1+\gamma(|\beta\setminus\alpha|)} \tag{3}$$

Next, to bound $\sum_{\alpha\in I} \sum_{\beta \in J_\alpha^*} \mathrm{E}\left[Y_\alpha Y_\beta\right]$, we partition J_α^*, into disjoint parts $U_1, U_2, \cdots, U_{2^k}$ according to the number of elements that differ from α. In other words, if $\beta \in U_i$, then $|\beta \setminus \alpha| = i$. For example, for a candidate solution $\alpha = (1, 2, 3, 4)$ (implying $k = 2$), U_1 contains $(1, 2, 3, 5)$, $(1, 6, 2, 3)$, etc. By a simple counting argument,

$$|U_i| = \binom{2^k}{2^k - i} \binom{N - 2^k}{i} \frac{(2^k)!}{2^{2^k - 1}} \tag{4}$$

For $\beta \in U_m$, we have derived an upper bound that $\mathrm{E}\left[Y_\alpha Y_\beta\right] \leq p^{2^k+1+\gamma(m)}$ in Equation 3. In Lemma 2, we would like to bound the number of candidate solutions in U_m that can reach this upper bound. To do so, we introduce some additional notations. For an integer $0 < m < 2^k$, write m as a sum of a powers of 2 in ascending order l_i, i.e., $m = \sum_{i=1}^{\zeta} 2^{l_i}$ where $0 \leq l_1 < l_2 < \cdots l_\zeta < k$.

Lemma 2. *Let $\hat{U}_m \subset U_m$ be the set of candidate solutions that achieve the maximum correlation $p^{2^k+1+\gamma(m)}$. $|\hat{U}_m| \leq \binom{N-2^k}{m} \cdot \frac{m!}{2^{m-\zeta}} \cdot \frac{2^k}{2^{l_1}}$.*

Proof. As mentioned, the maximum correlation appears when black nodes at the leaf label are closest to each other. We calculate how many such max-correlation configurations exist. First, the 2^{l_ζ} leaves of a certain subtree of depth l_ζ should be taken up by black nodes. There are $\frac{2^k}{2^{l_\zeta}}$ subtrees of depth l_ζ in total. After choosing one subtree of depth l_ζ, all the remaining black nodes should appear in the sibling subtree of depth l_ζ. Similarly, within that sibling subtree of depth l_ζ, a certain subtree of depth $l_{\zeta-1}$ should be taken by black nodes, giving $2^{l_\zeta - l_{\zeta-1}}$ possible ways. We

then repeat the above argument on the next subtree of depth $l_{\zeta-2}$ until we place all the m black nodes. Therefore, the total number of the candidate solutions in U_m that achieve the maximum correlation is at most

$$|\hat{U}_m| \leq \binom{N-2^k}{m} \cdot \frac{m!}{\prod_{i=1}^{\zeta} 2^{2^{l_i}-1}} \cdot \frac{2^k}{2^{l_\zeta}} \cdot \frac{2^{l_\zeta}}{2^{l_{\zeta-1}}} \cdot \ldots \cdot \frac{2^{l_2}}{2^{l_1}} = \binom{N-2^k}{m} \cdot \frac{m!}{2^{m-\zeta}} \cdot \frac{2^k}{2^{l_1}}$$

We can now finally bound the correlation sum in Lemma 1. While $\forall \beta \in \hat{U}_m$ achieves the maximum correlation by definition, $\forall \beta \notin \hat{U}_m$ will have a correlation that is at most p times the maximum, because its corresponding binary tree has at least one more black node. Therefore,

$$\begin{aligned}\sum_{\alpha \in I} \sum_{\beta \in J_\alpha^*} \mathrm{E}\left[Y_\alpha Y_\beta\right] &\leq |I| \cdot \Pr(Y_\alpha = 1) \sum_{\beta \in J_\alpha^*} \Pr(Y_\beta = 1 | Y_\alpha = 1) \\ &\leq \mathrm{E}\left[|L^{(k)}|\right] \cdot \sum_{i=0}^{2^k-1} \left[|\hat{U}_i| + p\left(|U_i| - |\hat{U}_i|\right)\right] p^{2^k+1+\gamma(i)}\end{aligned}$$

where $\mathrm{E}\left[|L^{(k)}|\right]$ is given in Theorem 1, $|\hat{U}_i|$ is given in Lemma 2, $|U_i|$ is given in Equation (4), $p = 2^{-\frac{n}{k+1}}$, and $\gamma(i)$ is defined in Equation (2).

Example numerical calculation. Suppose $k = 2$. For brevity, we temporarily write $\Pr(Y_\beta = 1 | Y_\alpha = 1)$ as $P_{\beta\alpha}$ for short. We have

- $|U_0| = \frac{4!}{2^3} = 3$, and $\forall \beta \in U_0$, $P_{\beta\alpha} \leq p$;
- $|U_1| = 3 \cdot \binom{N-4}{1} \cdot \binom{4}{1}$, $|\hat{U}_1| = 4(N-4)$;
- $|U_2| = 3 \cdot \binom{N-4}{2} \cdot \binom{4}{2}$, $|\hat{U}_2| = 2\binom{N-4}{2}$; and $\forall \beta \in \hat{U}_1 \cup U_2, P_{\beta\alpha} \leq p^3$.
- $|U_3| = |\hat{U}_3| = 3 \cdot \binom{N-4}{3} \cdot \binom{4}{3}$, and $\forall \beta \in U_3, P_{\beta\alpha} \leq p^4$.

Denote the right hand side in Lemma 1 as Δ. Plugging in a few example values, we have

- For $n = 30$ and $N = 2 \times 2^{\frac{n}{k+1}}$, $\sum_{\beta \in J_\alpha^*} P_{\beta\alpha} < 0.021$ and $\Delta < 0.037$;
- For $n = 100$ and $N = 2 \cdot 2^{\frac{n}{k+1}}$, $\sum_{\beta \in J_\alpha^*} P_{\beta\alpha} < 2.033 \cdot 10^{-9}$ and $\Delta < 3.511 \cdot 10^{-9}$;
- For a larger list size $N = 10 \cdot 2^{\frac{n}{k+1}}$ with $n = 100$, $\sum_{\beta \in J_\alpha^*} P_{\beta\alpha} < 1.942 \cdot 10^{-7}$ and $\Delta < 3.790 \cdot 10^{-9}$;

For a few more examples,

- For $k = 3$, $n = 100$, and $N = 4 \times 2^{\frac{n}{k+1}}$, $\Delta < 2.1 \times 10^{-3}$;
- For $k = 3$, $n = 120$, and $N = 5 \times 2^{\frac{n}{k+1}}$, $\Delta < 3.1 \times 10^{-4}$;
- For $k = 3$, $n = 120$, and $N = 10 \times 2^{\frac{n}{k+1}}$, $\Delta < 3.8 \times 10^{-2}$;
- For $k = 4$, $n = 200$, and $N = 4 \times 2^{\frac{n}{k+1}}$, $\Delta < 3.8 \times 10^{-5}$;
- For $k = 4$, $n = 250$, and $N = 5 \times 2^{\frac{n}{k+1}}$, $\Delta < 1.1 \times 10^{-6}$.

The above calculations show that the distribution of the number of solutions produced by Algorithm 1 can be closely approximated by a Poisson distribution. The total variation distance Δ between the two is small.

3 Learning Parity with Noise

3.1 Background

The Learning Parity with Noise (LPN) problem is a famous open problem that is widely conjectured to be hard. It forms the foundation of several primitives in lightweight cryptography and post-quantum cryptography. It is also a special case of the Learning With Error (LWE) problem, which has a reduction from the Shortest Independent Vector Problem (SIVP) [25] and has enabled numerous works in lattice-based cryptography [24,23].

Definition 3 (LPN). *Find the secret bit vector* $\mathbf{s} \in \mathbb{Z}_2^n$*, given samples in the form* $\{b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle \oplus e_i\}$ *where each* $\mathbf{a}_i \in \mathbb{Z}_2^n$ *is a random* n*-bit string, and each* $e_i \in \{0, 1\}$ *is a Bernoulli noise with parameter* $0 < \tau < 0.5$*.*

Starting from the seminal work by Blum, Kalai and Wasserman [6], LPN solving algorithms and heuristics largely follow the "reduce-and-solve" framework below.

- **The reduction phase**. Find a subset of samples $\{b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle \oplus e_i\}$ such that $\sum \mathbf{a}_i$ is one of the n bases of $\mathbb{Z}_2^n$. The most popular choice is the standard orthogonal bases, in which case the reduction phase becomes a subset sum problem. For brevity and without loss of the generality, we focus on the first bit of $\mathbf{s}$, denoted by s_1. The reduction phase looks for samples such that $\sum \mathbf{a}_i = (1, 0, \cdots, 0)$. Adding up the samples yield $\hat{b} = s_1 \oplus \hat{e}$ where $\hat{b} = \sum b_i$ and $\hat{e} = \sum e_i$. We call these output samples of the reduction phase *reduced samples*.
- **The solving phase**. With abundant reduced samples $\{\hat{b}\}$, solve s_1.

LPN solving algorithms/heuristics differ in the detailed strategies for the reduction phase and the solving phase. In all existing proposals we know of, the reduction phase always uses some type of iterative collision search procedure. The reduction phase of BKW in each step adds one sample to a set of other samples to cancel out a chunk of bits in $\sum \mathbf{a}_i$, and in the end obtains one reduced sample. BKW then repeats the collision search procedure on fresh samples to obtain more independent reduced samples.

For the reduction phase, the two most popular techniques are simple majority voting and Fast Walsh-Hadamard Transform. BKW uses simple majority voting: given abundant reduced samples $\{\hat{b}\}$, if there are more 0's than 1's, guess $s_1 = 1$; otherwise, guess $s_1 = 0$. Levieil and Fouque [17] proposed recovering multiple secret bits at a time in the solving phase and using the Fast Walsh-Hadamard Transform, which we explain in Section 3.4.

3.2 LPN Reduction Phase using Iterative Collision Search

The BKW algorithm only obtains one reduced sample from each run of the reduction phase in order to ensure independence among reduced samples to apply the Chernoff bound in the solving phase. As a result, BKW is extravagant in consuming input samples and does not mind "missing" many candidate reduced samples. Similar to the subset sum case, the single-list pair-wise iterative collision search, known as the LF2 method in the LPN literature, will produce far more reduced samples given the same amount of initial samples. The LF2 method has been an important technique, and has been adopted by every subsequent LPN solving work that we know of. But prior to our work, LF2 remains a heuristic with no rigorous analysis available. In particular, it remains open

after a decade how many reduced samples LF2 produces, to what degree these reduced samples are correlated, and to what extent the correlation affects the solving phase. We now answer these questions with rigorous analysis.

Although the reduction phase of LPN is almost exactly the same as a subset sum problem if we think of the vectors $\{\mathbf{a}_i\}$ as the bit-strings in the list L of subset sum, several remarks should be made regarding the collision schedule, i.e., how many bits to cancel at each step.

1. There is no agreed upon collision schedule in the literature. The original LF2 method [17] was inspired by Wagner's algorithm [26], which cancels out $\frac{2n}{k+1}$ bits in the last step and $\frac{n}{k+1}$ bits in every other step. Many subsequent works define LF2 to cancel out $\frac{n}{k}$ bits in every step including the last one. Our analysis will assume the original collision search schedule by Wagner, but can be extended to other schedules. With Wagner's schedule, our analysis for the number of solutions (both expectation and distribution) in Section 2 would apply if we only output fixed weighted reduced samples. But we note that it is OK for the LPN reduction phase to output reduced samples with weights lower than 2^k. So the total number of reduced samples will be greater than what our analysis in Section 2 indicates. We omit the analysis of this effect because more reduced samples improve the success rate of the solving phase.
2. The number of input samples to the reduction phase (i.e., the original list size $N = \left|L^{(0)}\right|$) greatly influences the expected number of reduced samples output by the reduction phase. If we set $N = 2 \times 2^{\frac{n}{k+1}}$ as in Section 2.4, then the list size at each step roughly remains the same (or slightly decreases) and the expected number of output samples is less than 2. However, in LPN, we would like the reduction phase to produce more samples for the solving phase. An easy way to achieve this is to increase the initial list size N to be slightly larger than $2 \times 2^{\frac{n}{k+1}}$. In this case, the list size will grow after every step before the last step.
3. Another way to obtain more reduced samples is to adjust the collision search schedule to cancel out slightly fewer than $\frac{2n}{k+1}$ bits in the last step, and slightly more than $\frac{n}{k+1}$ bits in every other step. The optimal collision schedule is outside the scope of this paper.
4. Bogos et al. [7] used an oversimplified combinatorial method to estimate the expected number of reduced samples, which led to the conclusion that $N = 3 \times 2^{\frac{n}{k+1}}$ would keep the list size constant across steps. Our analysis shows this is not true. Plugging into Equation (1), we can see that $N = 3 \times 2^{\frac{n}{k+1}}$ will cause the list size to grow exponentially after each step.
5. Another flaw in previous work is the LF(4) proposal by Zhang et al. [27]. It generalizes the LF2 method by with the intention to check all 4-tuple combinations instead of 2-tuple combinations. However, the scheme presented in [27] approximates the 4-tuple collision search using a 2-tuple collision search. This is essentially LF2 with the number of steps k doubled, and hence will not produce the claimed number of reduced samples. On the other hand, if a scheme really enumerates all 4-tuple combinations by brute force, the time complexity will become much more formidable than what's reported in [27], and it remains unclear whether the increased number of reduced samples can make up for it.

3.3 LPN Solving Phase with Majority Voting

This subsection and the next one analyze how the correlation between reduced samples affects the solving phase. Several previous works [10,27,7] have experimentally shown that the correlation does not seem to cause problems in the solving phase. Our analysis will provide theoretical support for these experimental results. We show the correlation between reduced samples produced by the

iterative collision search is weak and does not affect the success rate too much. This subsection focuses on the majority voting method, while the next subsection studies the fast Walsh-Hadamard transform method.

Recall that the majority voting method tallies the reduced samples $\{\hat{b}\}$, and guesses $s_1 = 1$ if there are more 1's than 0's, and guesses 0 otherwise. Since $\hat{b} = s_1 \oplus \hat{e}$, each $\hat{e} = 1$ contributes an incorrect vote. Define $Z_\alpha = Y_\alpha \hat{e}_\alpha$ where Y_α is defined in Section 2 and $\hat{e}_\alpha = \oplus_{i \in \alpha} e_i$. Let $W = \sum_{\alpha \in I} Z_\alpha$. W represents the number of incorrect votes among the reduced samples. If W does not exceed one half of the reduced samples, then the majority voting will guess s_1 correctly.

If $\{Z_\alpha\}$ were independent, a Chernoff bound would suffice like in BKW [6]. The main difficulty we face is to bound $\Pr(W \geq w)$ when $\{Z_\alpha\}$ are not independent. We will show that if we calculate this bound pretending that $\{Z_\alpha\}$ are independent, the error will be very small.

Let W' be the sum of $|I|$ independent Bernoulli random variables (cf. the definition of W). Each addend Z' follows the same distribution as Z_α, i.e., $\Pr(Z' = 1) = \Pr(Z_\alpha = 1) = \Pr(Y_\alpha = 1) \Pr(\hat{e}_\alpha = 1)$. We once again invoke the Chen-Stein method [1] to bound the total variation distance between W and W',

$$\Delta' = \sum_{l=0}^{\infty} |\Pr(W = l) - \Pr(W' = l)|.$$

We introduce an intermediate random variable Π that follows a Poisson distribution with mean $\lambda' = \mathrm{E}[W]$. Using the triangle inequality, we have $\Delta' \leq \Delta'_1 + \Delta'_2$ where Δ'_1 and Δ'_2 are the total variation distances between W and Π, and between Π and W', respectively. Δ'_1 can be bounded in the same way as in Section 2.5. Recall that $W = \sum_{\alpha \in I} Z_\alpha$, $|L^{(k)}| = \sum_{\alpha \in I} Y_\alpha$ and $Z_\alpha = Y_\alpha \hat{e}_\alpha \leq Y_\alpha$. So Δ'_1 is no larger than Δ.

W' follows a binomial distribution, which is frequently approximated by a Poisson distribution. Concretely, we can bound their total variation distance using the Chen-Stein method. Note that for each addend Z' of W', the neighborhood of dependence of Z' is empty, so only the first double sum in the Chen-Stein method (cf. Lemma 1) remains.

$$\Delta'_2 = \sum_{l=0}^{\infty} |\Pr(W' = l) - \Pr(\Pi = l)| \leq \frac{4(1 - e^{\lambda'})}{\lambda'} \cdot |I| \cdot \big(\Pr(Z' = 1)\big)^2$$

Observe that $\lambda' = \mathrm{E}[W'] = |I| \cdot \Pr(Z' = 1)$, $\Pr(Z' = 1) = \Pr(Y_\alpha = 1) \cdot \Pr(\hat{e}_\alpha = 1)$, $\Pr(Y_\alpha = 1) = p^{2^k}$, and $\Pr(\hat{e}_\alpha = 1) = \Pr(\oplus_{i \in \alpha} e_i = 1) = \frac{1 - (1-2\tau)^{2^k}}{2}$ [6]. Thus,

$$\Delta'_2 \leq 4(1 - e^{\lambda'}) \cdot \Pr(Z' = 1) \leq 4(1 - e^{\lambda'}) \cdot p^{2^k} \cdot \frac{1 - (1-2\tau)^{2^k}}{2} < 2p^{2^k}.$$

Clearly, Δ'_2 is very small compared to Δ'_1, so $\Delta' \approx \Delta'_1 \leq \Delta$.

W' is a sum of independent Bernoulli random variables, so the Chernoff bound can be applied to $\Pr(W' \geq w)$. $\Pr(W \geq w)$ can then be bounded by $\leq \Pr(W' \geq w) + \Delta'$. This means the correlation between votes (i.e., reduced samples) resulting from the reduction phase lowers the success rate by at most Δ compared to independent votes. Section 2.5 has shown that Δ is very small, ranging from 0.02 to 10^{-9}. This explains why previous works observed that majority voting using correlated reduced samples works well in reality.

3.4 LPN Solving Phase with Fast Walsh-Hadamard Transform

Levieil and Fouque [17] proposed applying the Fast Wash-Hadamard Transform (FWHT) and recovering a block of secret bits at a time. They call this method LF1. We describe the LF1 method below.

Since LF1 tries to recover a block of n' secret bits at a time, it needs to modify the reduction phase to generate reduced samples that depend on n' bits of the secret. This is a straightforward modification that simply involves cancelling out fewer bits ($n - n'$ instead of $n - 1$). Denote these reduced samples as $\hat{b}_l = \langle \hat{a}_l, s \rangle \oplus \hat{e}_l$ where $\hat{a}_l, s \in Z_2^{n'}$, i.e., we focus on the n' secret bits we are trying to guess.

In the solving phase, for $x \in \{0,1\}^{n'}$ define $f(x) = \sum_l \delta(a_l, x)(-1)^{b_l}$ where $\delta(a_l, x) = 1$ if $a_l = x$ and 0 otherwise. LF1 applies FWHT to compute for each $v \in \{0,1\}^{n'}$,

$$\hat{f}(v) = \sum_x (-1)^{\langle x,v \rangle} f(x) = \sum_l (-1)^{\langle \hat{a}_l, s \oplus v \rangle \oplus \hat{e}_l} \tag{5}$$

Observe that $\hat{f}(s) = \sum_l (-1)^{\hat{e}_l}$. Since $\Pr(\hat{e}_l = 0) > \Pr(\hat{e}_l = 1)$, $f(s)$ should be noticeably larger than 0. On the other hand, for $s' \neq s$, $e'_l = \langle \hat{a}_l, s' \oplus s \rangle$ is uniformly random, and $f(s')$ should be close to 0. LF1 then picks the largest $\hat{f}(v)$ and guesses $s = v$. Thus, if there exists $s' \neq s$ such that $\hat{f}(s') \geq \hat{f}(s)$, then the LF1 method fails. For each s', the probability that $\hat{f}(s') \geq \hat{f}(s)$ is

$$\epsilon = \Pr\left(\hat{f}(s') \geq \hat{f}(s)\right) = \Pr\left(\sum_l e'_l \leq \sum_l \hat{e}_l\right) \tag{6}$$

When analyzing the success rate of LF1, there are two places that prior works argue heuristically [17,10,27,7,8]. One is that they assume reduced samples are independent. The other one is that after noting LF1's success requires $\forall s' \in \{0,1\}^{n'}$, $\hat{f}(s') < \hat{f}(s)$, they assume independence between these events and approximate the success rate of LF1 as $(1-\epsilon)^{2^{n'}-1}$.

We now present a rigorous analysis for LF1's success rate. Note that the second inaccuracy above can be easily fixed by a union bound: $\Pr(\text{LF1 succeeds}) \geq 1 - 2^{n'}\epsilon$. So it remains to bound ϵ. The difficulty again lies in analyzing $\sum_l \hat{e}_l$ for correlated $\{\hat{e}_l\}$. We use similar techniques as before. Write $S = \sum_l \hat{e}_l = \sum_{\alpha \in I} Y_\alpha \hat{e}_\alpha$ and $T = \sum_l e'_l = \sum_{\alpha \in I} Y_\alpha \cdot \langle \hat{a}_\alpha, s' \oplus s \rangle$. Define S' to be the sum of I independent Bernoulli random variables each with mean $\Pr(Y_\alpha \hat{e}_\alpha = 1) = \frac{1}{2} \cdot p^{2^k} \cdot (1 - (1-2\tau)^{2^k})$. Define T' to be the sum of I independent Bernoulli random variables each with mean $\Pr(e'_l = 1) = \frac{1}{2}$. We again have $\sum_{l=0}^{\infty} |\Pr(S = l) - \Pr(S' = l)| \leq \Delta'_3 \leq \Delta$ and $\sum_{l=0}^{\infty} |\Pr(T = l) - \Pr(T' = l)| \leq \Delta'_3 \leq \Delta$. Therefore,

$$\epsilon = \Pr(T \leq S) \leq \Pr(T' \leq S') + 2\Delta'_3 \leq \Pr(T' \leq S') + 2\Delta.$$

Now T' and S' are sums of independent of random variables, so $\Pr(T' \leq S')$ can be bounded using the central limit theorem or the Hoedffing bound. We omit these details as several prior works [27,**?**,8] have included such analysis. Again, this means a heuristic estimation of the success rate by pretending that T and S are sums of independent random variables is only off by at most 2Δ, which is very small. This shows the Fast Walsh-Hadamard Transform method for the solving phase has good success rate under suitable parameters.

4 Conclusion

Iterative collision search is a crucial technique in solving subset sum and LPN. The single-list pair-wise variant has so far been the most efficient variant for random fixed weighted subset sum and LPN, but has not been rigorously analyzed prior to our work. In this paper, we presented rigorous analysis for the single-list pair-wise iterative collision search procedure and its applications in random fixed weighted subset sum and LPN. In the LPN context, we show that while the reduced samples produced by this method are correlated, the correlation is weak and barely decreases the success rate of LPN solving. Our analysis of the single-list pair-wise iterative collision search is also applicable to LWE. It remains interesting future work to study how it interacts with other techniques in the LWE literature.

References

1. 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.
2. Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A family of fast syndrome based cryptographic hash functions. In *International Conference on Cryptology in Malaysia*, pages 64–83. Springer, 2005.
3. Daniel J Bernstein. Better price-performance ratios for generalized birthday attacks. In *Workshop Record of SHARCS*, volume 7, page 160, 2007.
4. Daniel J Bernstein, Tanja Lange, Ruben Niederhagen, Christiane Peters, and Peter Schwabe. Fsbday. In *International Conference on Cryptology in India*, pages 18–38. Springer, 2009.
5. Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. In *NDSS*, 2016.
6. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, 2003.
7. Sonia Bogos, Florian Tramer, and Serge Vaudenay. On solving LPN using BKW and variants. *Cryptography and Communications*, 8(3):331–369, 2016.
8. Sonia Bogos and Serge Vaudenay. Optimization of lpn solving algorithms. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 703–728. Springer, 2016.
9. Henri Gilbert, Matthew JB Robshaw, and Yannick Seurin. Hb#: Increasing the security and efficiency of hb+. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 361–378. Springer, 2008.
10. Qian Guo, Thomas Johansson, and Carl Löndahl. Solving lpn using covering codes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2014.
11. Nicholas J Hopper and Manuel Blum. Secure human identification protocols. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 52–66. Springer, 2001.
12. Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 235–256. Springer, 2010.
13. Ari Juels and Stephen A Weis. Authenticating pervasive devices with human protocols. In *Annual International Cryptology Conference*, pages 293–308. Springer, 2005.
14. Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
15. Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient authentication from hard learning problems. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 7–26. Springer, 2011.
16. Paul Kirchner. Improved generalized birthday attack. Cryptology ePrint Archive, Report 2011/377, 2011.
17. Éric Levieil and Pierre-Alain Fouque. An improved lpn algorithm. In *International Conference on Security and Cryptography for Networks*, pages 348–359. Springer, 2006.
18. Vadim Lyubashevsky. On random high density subset sums. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 12. Citeseer, 2005.
19. Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 378–389. Springer, 2005.

20. Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. *Automata, Languages and Programming*, pages 144–155, 2006.
21. Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. Swifft: A modest proposal for fft hashing. In *International Workshop on Fast Software Encryption*, pages 54–72. Springer, 2008.
22. Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 586–595. SIAM, 2009.
23. Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.
24. Krzysztof Pietrzak. Cryptography from learning parity with noise. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 99–114. Springer, 2012.
25. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.
26. David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.
27. Bin Zhang, Lin Jiao, and Mingsheng Wang. Faster algorithms for solving lpn. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 168–195. Springer, 2016.