New Algorithms for Solving LPN

Bin Zhang[†,‡] and Xinxin Gong[†]

[†] TCA Laboratory, SKLCS, Institute of Software, Chinese Academy of Sciences
[‡] State Key Laboratory of Cryptology, P.O.Box 5159, Beijing, 100878, China
{zhangbin, gongxinxin}@tca.iscas.ac.cn

Abstract. The intractability of solving the LPN problem serves as the security source of many lightweight/post-quantum cryptographic schemes proposed over the past decade. There are several algorithms available so far to fulfill the solving task. In this paper, we present further algorithmic improvements to the existing work. We describe the first efficient algorithm for the single-list k-sum problem which naturally arises from the various BKW reduction settings, propose the hybrid mode of BKW reduction and show how to compute the matrix multiplication in the Gaussian elimination step with flexible and reduced time/memory complexities. The new algorithms yield the best known tradeoffs on the complexity curve and clearly compromise the 80-bit security of the LPN instances suggested in cryptographic schemes. Practical experiments on reduced LPN instances verified our results.

Keywords: LPN, Single-list k-sum problem, Gaussian elimination, Tradeoff, BKW.

1 Introduction

The problem of Learning Parity with Noise (LPN) has found many interesting applications in modern cryptography [2, 14, 15, 17, 19–23, 25, 28, 32], whose search version can be converted into the well-studied NP complete problem of decoding a binary random linear code in coding theory. Besides, this problem also lies at the central of learning theory, i.e., an algorithm for solving LPN could be used to learn several important concept classes, e.g., 2-DNF formulas, juntas and any function with a sparse Fourier spectrum [13].

The fastest known algorithms for solving LPN runs in sub-exponential time in the general case and in contrast to most of the number-theoretic problems in cryptography, there is no available quantum algorithms that could solve LPN in a significantly faster time than the classical ones. Further, the decisional version of the LPN problem is polynomial equivalent to the search variant [8, 24] and there is no faster algorithms for decisional than for search.

The LPN based schemes often enjoy the extreme simplicity and efficiency, making themselves promising candidates for various applications. There are many cryptographic primitives that could be constructed from LPN. One can build a one-way function (OWF) from LPN and from a OWF, pseudorandom

generators/functions/permutations can be constructed accordingly [17, 19, 28], e.g, it is shown how to constructed a linear-stretch pseudorandom generator from LPN in [2]. In [15], a simple secret-key encryption scheme based on LPN is proposed, which is provably secure not only in the standard sense, but also in more interesting circumstances. There are also quite a few LPN based secret-key identification protocols, e.g., HB, HB$^+$, HB$^\#$ and AUTH authentication protocols as well as message authentication code (MAC) [14, 21, 23, 25]. What's more, efficient zero-knowledge proof of knowledge can be obtained by modifying Stern's protocol and string commitment schemes can be built from LPN [22, 32].

Undoubtedly, mounting attacks on generic LPN is a very important line of work. The seminal algorithm proposed by Blum et al. in [7], known as the BKW algorithm, employs an iterated collision procedure of the queries to reduce the dependency on the information bits at the expense of a folded noise level. Later in [27], it is suggested to exploit the Fast Walsh-Hadamard Transform (FWHT) in the secret solving phase, with an analysis of different security levels achieved by different LPN instances, which are referenced by most of the work thereafter. In [26], Kirchner suggested to transform the problem into a systematic form, where each secret bit appears as an observed symbol perturbed by noise. In [6], it is demonstrated by Bernstein and Lange how to utilize the ring structure of Ring-LPN in matrix inversion to further reduce the attack complexity, which is applicable to the common LPN instances with a slight modification as well. Then the first attack compromising the 80-bit security of the $(512, 1/8)$-LPN instance came. At Asiacrypt 2014, a new algorithm using covering code was presented [18], which broke the 80-bit security bound with a complexity of $2^{79.7}$. At Eurocrypt 2016, faster algorithms for solving LPN were proposed in [34] based on an optimal precise embedding of cascaded perfect codes with the parameters found by integer linear programming to efficiently reduce the dimension of the secret information. This new technique is claimed to be generic and can be applied to any given (k, η)-LPN instance, while in [18] the code construction methods for covering are missing and only several specific parameters for the $(512, 1/8)$-LPN instance were given in their conference presentation. However, the results in [34] are debated by [9] and some inconsistency in the theoretical analysis of [34] are discovered and reported. Recently, a new algorithm is proposed in [10] by trying to make the best use of the existing techniques and automatize the generation of LPN solving algorithms from the considered parameters. Specifically, a LPN algorithm is formalized as a path in a graph and the solving algorithm is searching for the optimal paths in the graph. It is reported that the common $(512, 1/8)$-LPN instance can be broken within a complexity of $2^{79.37}$ following their analysis in [10].

Our Contributions. In this paper, we present further algorithmic improvements on this problem to address the unsolved issues, clarify the controversial points and provide the complexity estimates comprehensively. First, when[1] k is a power of 2, an efficient algorithm for solving the single-list k-sum problem is

[1] In the paper, the single-list k-sum, LF(k) and the (k, η)-LPN instance share a common notation k, it is easy to distinguish them from each other in the context.

described, which gives an answer to the open problem of Minder and Sinclair in [31]. Without dividing the whole effective list into several sub-lists, we remould Wagner's k-tree algorithm in [33] for the multi-list k-sum problem by virtualizing most of the processing tree. This algorithm is naturally connected to the LF(k) reduction in [34] and exhibits a tradeoff between the list size and the running time. It is shown that as long as a solution exists with a reasonable probability, our algorithm will eventually find it out with a low complexity and significantly smaller list size than Wagner's algorithm. For completeness, the generalized version of the algorithm is also developed to cover the non-binary alphabet case, which may be of independent interest. Second, based on the single list k-sum algorithm, we suggest some hybrid reduction modes to deal with different attack conditions and requirements flexibly, e.g., LF(4) + LF2, LF(8) + LF2 and LF(16) + LF2, which are to be utilized in the collision procedure of the LPN solving algorithm. Opposite to the previous relevant algorithms which only adopt one fixed sort of reduction, the new hybrid mode can make good use of different kinds of reductions without increasing the overall complexity and achieve better tradeoffs which are impossible for the old strategies. Third, we show how to accomplish the dominant task of matrix multiplication in the Gaussian elimination step of the LPN solving algorithms with considerably reduced time/memory complexities, compared to the methods suggested in [5, 9, 34]. The idea is to replace the large look-up table used by the algorithm in [34] with a relatively small and lightweight table, and determine the desired value in a deliberately order by reading this small table continually. This is illustrated by both concrete LPN instances and theoretical analysis. Last, we provide a unified framework for solving LPN by integrating the new methods with the theoretical predictions of all the involved phases and aspects, and apply it to the three core LPN instances, $(512, \frac{1}{8})$, $(532, \frac{1}{8})$, $(592, \frac{1}{8})$-LPN instances respectively. There are explicit improvement factors achieved by the new algorithms, compared with the previously known best results in [18][2] and [10][3], shown in Table 1. Please see Table 10 in Section 8.1 for more tradeoff choices and more complexity aspects. We have implemented the new attacks in practice on reduced LPN instances and the theoretical results have been verified in experiments.

This paper is organized as follows. We first introduce some preliminaries of the LPN problem that are relevant to our work in Section 2 with a necessary review of the LPN solving algorithms in [10, 18, 34]. Next in Section 3, we develop the algorithm to resolve the single-list k-sum problem, which is further generalized in Section 4 to the non-binary alphabet cases, and present some LF(k)

[2] We choose $8l \ln 2/\varepsilon_f^2$ as the number of queries needed in the final solving phase to assure a success probability of almost 1, which is twice as that presented in the presentation at Asiacrypt 2014 [18]. If we choose the data complexity as theirs for comparison, our complexity can further be reduced. Besides, the complexity $2^{82.27}$ on the second column is a corrected result by [10] for the complexity presented at Aisacrypt 2014, and the time $2^{89.04}$ is a corrected result for the complexity in the publication at Aisacrypt 2014.

[3] The number of queries we choose can assure a success probability of almost 1, while the number in [10] is set to be with a probability of failure upper bounded by 33%.

Table 1: Comparison of the time complexities of different LPN solving algorithms

(k,η)-LPN	Algorithm					
	[18]	[10]	**ours**	**ours**	**ours**	**ours**
$(512, 1/8)$	$2^{82.27}$	$2^{79.37}$	$2^{74.75}$	$2^{74.30}$	$2^{74.83}$	$2^{73.92}$
$(532, 1/8)$	$2^{90.43}$	$2^{81.64}$	$2^{76.86}$	$2^{76.42}$	$2^{76.94}$	$2^{75.99}$
$(592, 1/8)$	$2^{97.87}$	$2^{88.25}$	$2^{83.79}$	$2^{83.60}$	$2^{82.89}$	-

schemes for $k = 4, 8, 16$. Based on it, we propose the hybrid reduction modes with the concrete depictions of LF(4) + LF2, LF(8) + LF2 and LF(16) + LF2 in Section 5. Then in Section 6, we present the crucial improvements on matrix multiplication in the Gaussian elimination procedure of the LPN solving algorithms. In Section 7, we present the theoretical complexity analysis of our improved algorithm by integrating the new methods into a new framework. The applications to the three core LPN instances are given in Section 8 with the experimental results. Finally, some conclusions are made in Section 9.

2 Preliminaries

In this section, some notations and basic definitions relevant to our work are presented, together with a detailed review of the previous LPN solving algorithm in [34] and the necessary coverings of those in [10, 18].

2.1 The Search LPN Problem

For simplicity, we follow the conventional notations and descriptions in this domain. The vectors and matrices are denoted by small and capital bold letters, respectively. For a distribution D and a random variable x, $x \leftarrow D$ denotes that x is sampled according to D. For a set $\mathcal{X}$, $x \xleftarrow{\$} \mathcal{X}$ denotes that x is assigned a value sampled uniformly at random from $\mathcal{X}$. The Bernoulli distribution with parameter η is denoted by $\mathtt{Ber}_\eta$, i.e., if $e \leftarrow \mathtt{Ber}_\eta$ then $\Pr[e = 1] = \eta$ and $\Pr[e = 0] = 1 - \eta$. The binary field is denoted by GF(2) and the scalar/inner product of two m-dimensional vectors $\mathbf{a}, \mathbf{b} \in \mathrm{GF}(2)^m$ is defined as $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a} \cdot \boldsymbol{b}^T = \bigoplus_{i=0}^{m-1} a_i b_i$, where $\mathrm{GF}(2)^m$ is the m-dimensional vector space over GF(2) and $\boldsymbol{b}^T$ denotes the transpose of $\boldsymbol{b}$. The usual bitwise XOR is denoted by $+$.

Definition 1. *For a secret random vector $\boldsymbol{x} \in GF(2)^k$, an LPN oracle $\Theta_{\mathtt{LPN}_{k,\eta}}$ with a noise parameter $\eta \in (0, \frac{1}{2})$ returns independent samples of the form*

$$(\boldsymbol{g} \xleftarrow{\$} GF(2)^k, e \leftarrow \mathtt{Ber}_\eta : \langle \boldsymbol{x}, \boldsymbol{g} \rangle + e).$$

The search $\mathtt{LPN}_{k,\eta}$ problem consists of recovering the vector $\boldsymbol{x}$ according to the samples output by the oracle $\Theta_{\mathtt{LPN}_{k,\eta}}$. An algorithm $\mathcal{S}$ is called (n, t, m, δ)-solver

if $\Pr[S = \boldsymbol{x} : \boldsymbol{x} \xleftarrow{\$} \{0,1\}^k] \geq \delta$*, and it runs in time at most* t *and memory at most* m *with at most* n *oracle queries.*

As stated before, the search LPN problem can be converted into the well-studied problem of decoding a binary random linear code, thus it can be rewritten in matrix form as $\boldsymbol{z} = \boldsymbol{x}\mathbf{G} + \boldsymbol{e}$, where $\boldsymbol{x}$ is the k-bit secret vector and $\mathbf{G}$ is a $k \times n$ matrix formed as $\mathbf{G} = [\boldsymbol{g}_1^T, \boldsymbol{g}_2^T, ..., \boldsymbol{g}_n^T]$ with g_i^T being the ith column of $\mathbf{G}$.

It seems that the current state of the art of the methods for decoding such random binary linear codes [3, 4, 29, 30] have a complexity higher than the dedicated algorithms tailored for solving LPN directly, e.g., for the $(512, 1/8)$-LPN instance, the fastest decoding algorithm in [30] will have a time complexity of 2^{259} in the full distance decoding, given the codeword of length $2^{11.3828} = \frac{8 \cdot 512 \cdot \ln 2}{1 - H(1 - \frac{1}{8})}$ with $H(\cdot)$ being the binary entropy function. Therefore, we focus on such dedicated algorithms throughout this paper. Since both $\boldsymbol{e} = [e_1, e_2, ..., e_n]$ and $\boldsymbol{z} = [z_1, z_2, ..., z_n]$ are n-bit vectors, we have $z_i = \langle \boldsymbol{x}, \boldsymbol{g}_i \rangle + e_i$ holds for each $i = 1, 2, ..., n$. In principle, the search LPN problem asks to recover the secret vector $\boldsymbol{x}$, given a number of queries consisting of uniformly and randomly distributed vectors and the noisy scalar product outputs.

2.2 Review of the Previous LPN Solving Algorithms

In [10, 18, 34], three new LPN solving algorithms are proposed. In fact, these algorithms share many of the common points: the spirit and the general framework are the same, with only some algorithmic differences in the concrete steps. Since our refined framework for solving LPN is extracted mainly from the algorithm in [34], we will put an emphasis on that algorithm with some considerations to those in [10, 18] whenever necessary. Precisely, given a (k, η)-LPN instance, the algorithm from [34] performs the following steps in a sequential order.

Step 0. Sample selection. This is a reduction phase which transfers the (k, η)-LPN problem into a $(k - c, \eta)$-LPN problem. It keeps from $\Theta_{\mathtt{LPN}_{k,\eta}}$ only those samples that the last c bits of $\boldsymbol{g}$ equal 0 from the initial queries. Let N be the number of initial queries, then the complexity of this step is about $C_0 = N$. Note that there are an expected number of $n = N2^{-c}$ queries remained after this step, and the parameter k is changed to $k - c$.

Step 1. Gaussian elimination. Similar to that in [18], this step is to systematize $\mathbf{G}$ to change the distribution of the secret bits in $\boldsymbol{x}$ without changing the associated noise level. Let $\hat{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1} + [z_1 \ z_2 \ \cdots \ z_k]$, then $\hat{\mathbf{z}} = \hat{\mathbf{x}}\hat{\mathbf{G}} + \mathbf{e}$, where $\hat{\mathbf{G}} = \mathbf{D}\mathbf{G} = [\mathbf{I} \ \hat{\mathbf{g}}_{k+1}^T \ \hat{\mathbf{g}}_{k+2}^T \ \cdots \ \hat{\mathbf{g}}_n^T]$. The dominant part is to calculate $\mathbf{D}\mathbf{G}$.

In [18], the multiplication of $\mathbf{D}\mathbf{G}$ is optimized by using look-up tables. For a fixed a, divide the matrix $\mathbf{D}$ into a parts, i.e., $\mathbf{D} = [\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_a]$, where $\mathbf{D}_i$ is a sub-matrix with $s = \lceil \frac{k}{a} \rceil$ columns (except possibly the last matrix $\mathbf{D}_a$). All the possible values of $\mathbf{D}_i \boldsymbol{x}^T$ are stored for $\boldsymbol{x} \in \mathrm{GF}(2)^s$ indexed by $i = 1, ..., a$. The cost of constructing these a tables is $k^2 2^s$. For each column $\boldsymbol{g}^T$ of $\mathbf{G}$, $\mathbf{D}\boldsymbol{g}^T$ is written as $\mathbf{D}\boldsymbol{g}^T = \mathbf{D}_1 \boldsymbol{g}_1^T + \mathbf{D}_2 \boldsymbol{g}_2^T + ... + \mathbf{D}_a \boldsymbol{g}_a^T$, then $\mathbf{D}_i \boldsymbol{g}_i^T$ can be obtained by reading the table indexed by i for $i = 1, ..., a$, and further $\mathbf{D}\boldsymbol{g}^T$ can be got after

a calculation of the addition of a k-bit vectors. The complexity of this step is $(n-k)(a+(a-1)k)$.

In [34], several improvements on this step are presented. First, the procedure of constructing each table $\mathbf{D}_i\boldsymbol{x}^T$ for $\boldsymbol{x} \in \mathrm{GF}(2)^s$ can be improved by computing the products in a certain order with a reduced time complexity of $PC_{11} = (2^s - s - 1)ka$ for constructing a tables, which is much lower than $k^2 2^s$. The memory needed for storing the tables $[\boldsymbol{x}, \mathbf{D}_i\boldsymbol{x}^T]_{\boldsymbol{x}\in\mathrm{GF}(2)^s}$, $i = 1, ..., a$ is $M_{11} = 2^s(s+k)a$. Next, the addition of a k-bit vectors can be further optimized as follows. It is shown that the cost for calculating all the possible additions of q u-bit vectors is $\sum_{i=2}^{q} u2^{u-1}\binom{2^u-2}{i-1}$ in [34]. Based on this, the addition of a k-bit vectors is optimized through a second table look-up. For a properly chosen integer u and $d = \lceil \frac{k}{u} \rceil$, a table storing all the possible additions of a u-bit vectors is constructed, thus the adversary could read it d times to obtain the sum of a k-bit vectors. Thus the complexity of Step 1 can be reduced to $C_1 = (n-k)(a+d)$, which is much lower than the original complexity $(n-k)(a+(a-1)k)$. The complexity for constructing the table storing all the possible additions of a u-bit vectors is $PC_{12} = \sum_{i=2}^{a} u2^{u-1}\binom{2^u-2}{i-1}$ in time and $M_{12} = \sum_{i=2}^{a} \frac{i+1}{i} u2^{u-1}\binom{2^u-2}{i-1}$ in memory.

Step 2. Collision procedure. This step exploits the BKW reduction to make the length of the secret vector shorter. The reduction modes of LF1, LF2 and LF(4) are adopted in [34]. Denote $n[i]$, $i = 1, 2, ..., t$, as the expected number of samples via the i-th BKW step, where $n[0] = n - k$, $n[t] = m$ and m is the number of queries required for the final solving phase.

For LF1, suppose b bits are eliminated at each BKW step, then $n[i] = n - k - i2^b$. For the merging procedure, first sort the $n[i-1]$ samples to partition them into 2^b parts in term of the last b entries of each column, then choose a representative in each partition and add it to the rest of the samples in the same partition. Hence $n[i] = n[i-1] - 2^b$ solutions are obtained. To compute the sum of $n[i]$ pairs of $(k+1-ib)$-bit columns, just divide each pair into $\left\lceil \frac{k+1-ib}{f} \right\rceil$ parts, and read the sum of each part directly from a table storing all the possible additions of two f-bit vectors similarly as that in the Gaussian elimination step. Totally, t steps of LF1 will have a cost of[4] $C_2 = \sum_{i=1}^{t} \left(\left\lceil \frac{k+1-ib}{f} \right\rceil n[i] + n[i-1] \right)$, and also a pre-computation of $PC_2 = f2^{f-1}(2^f - 2)$ in time and $M_2 = \frac{3}{2} f2^{f-1}(2^f - 2)$ in memory, where f is a properly chosen integer.

For LF2, different from the method in LF1, now any pair in each partition is considered, and thus $n[i] = \binom{n[i-1]}{2} 2^{-b}$. Just do the same merging as LF1, and t steps of LF2 will have a cost of $C_2 = \sum_{i=1}^{t} \left(\left\lceil \frac{k+1-ib}{f} \right\rceil n[i] + n[i-1] \right)$, and also a pre-computation of $PC_2 = f2^{f-1}(2^f - 2)$ in time and $M_2 = \frac{3}{2} f2^{f-1}(2^f - 2)$ in memory.

[4] There is a term representing the sorting process at each BKW step added, which is $n[i-1]$ for $i = 1, ..., t$.

Finally for LF(4)[5], first try to find a number of 4-tuples that add to 0 in the last b entries. This is the same as the 4-sum problem. According to Wagner's k-tree algorithm [33], we have $n[i-1] = (4!)^{1/4}(2^b n[i])^{1/3}$ and the cost for finding $n[i]$ solutions is about $6 \times (2^b n[i])^{1/3}$, together with a space complexity of $3 \times (2^b n[i])^{1/3}$. Still do the same merging as LF1. In total, t steps of LF(4) will have a cost of $C_2 = \sum_{i=1}^{t} \left(\left\lceil \frac{k+1-ib}{f} \right\rceil n[i] + 6(2^b n[i])^{1/3} \right)$, and also a pre-computation of $PC_2 = \sum_{i=2}^{4} f2^{f-1} \binom{2^f-2}{i-1}$ in time and $\sum_{i=2}^{4} \frac{i+1}{i} f2^{f-1} \binom{2^f-2}{i-1}$ in memory storing all the possible additions of four f-bit vectors. Totally, the memory needed is $M_2 = \sum_{i=2}^{4} \left(\frac{i+1}{i} f2^{f-1} \binom{2^f-2}{i-1} + 3(2^b n[i])^{1/3} \right)$.

After this step, the bias of the noise is sharply reduced from the original bias $\varepsilon = 1 - 2\eta$ to $\varepsilon_{cp} \triangleq \varepsilon^{2^t}$ for LF1 and LF2, and to $\varepsilon_{cp} \triangleq \varepsilon^{4^t}$ for LF(4).

Step 3. Partial secret guessing. This step further guesses k_1 bits of the secret. More precisely, it tries all the possible values of the k_1-bit vectors whose Hamming weights are smaller than or equal to w_1. The complexity of this step is $C_3 = \sum_{i=0}^{w_1} \binom{k_1}{i}$.

Step 4. Covering-coding. A linear covering code is used in this step to further decrease the dimension of the secret vector. Different from that in [18], the explicit code constructions of the optimal cascaded perfect codes are provided in [34], from which the exact bias introduced in this step, denoted by ε_{cc}, is optimally derived from the integer programming. Consequently, this step will have a cost of $C_4 = mh$, where h is the number of chunks cascaded to construct the corresponding covering code, and also a pre-computation of $PC_4 = 11 \cdot 23 \cdot 2^{23}$ in time and $M_4 = (23 + 12) \cdot 2^{23}$ in memory[6].

Step 5. Subspace hypothesis testing. This step deals with the solving phase, consisting in applying the FWHT to recover the remaining l secret bits. The complexity of this step is $C_5 = l2^l \sum_{i=0}^{w_1} \binom{k_1}{i}$.

Complexity analysis. The final noise bias is $\varepsilon_f = \varepsilon_{cp}\varepsilon_{cc}$, which is $\varepsilon_f = \varepsilon^{2^t}\varepsilon_{cc}$ for LF1 and LF2, and $\varepsilon_f = \varepsilon^{4^t}\varepsilon_{cc}$ for LF(4). As illustrated in [34], it needs $m = 8l \ln 2/\varepsilon_f^2$ queries to distinguish the correct guess from the others in the final solving phase. Then the number of queries when adopting LF1 at Step 2 is $n = m + k + t2^b$, and this number when adopting LF2 is computed as $n[i-1] \approx (2!)^{1/2} n[i]^{1/2} 2^{b/2}$, $i = t, t-1, ..., 1$, where $n[t] = m$ and $n[0] = n-k$. For adopting LF(4), the number of queries should be computed as $n[i-1] = (4!)^{1/4}(2^b n[i])^{1/3}$, $i = t, t-1, ..., 1$. Once knowing n, the number of initial queries is computed as $N = n2^c$. In addition, the online time complexity is

$$C = C_0 + \frac{PC_{11} + C_1 + C_2 + C_3 + C_4 + C_5}{\Pr(w_1, k_1)},$$

[5] The calculation of the required number of queries for $n[i]$ solutions to exist and the complexity results, have been improved according to Wagner's k-tree algorithm. Specifically, we derive that $n[i-1] = (4!)^{1/4}(2^b n[i])^{1/3}$ and the time/space complexities for finding $n[i]$ solutions are $6 \times (2^b n[i])^{1/3}$ and $3 \times (2^b n[i])^{1/3}$, respectively.

[6] This complexity can be further reduced to $M_4 = 12 \cdot 2^{23}$, though it has no obvious effect on the whole complexity.

where $\Pr(w_1, k_1) = \sum_{i=0}^{w_1} \binom{k_1}{i}(1-\eta)^{k_1-i}\eta^i$, together with a pre-computation of $Pre = PC_{12} + PC_2 + PC_4$, and the memory needed is about $M = nk + M_{11} + M_{12} + M_2 + M_4$.

3 The Single List k-sum Problem

In this section, we will develop an algorithm for solving the single-list k-sum problem over GF(2), connect LF(k) with both the single-list and multi-list k-sum problem, and present some LF(k) schemes for $k = 4, 8, 16$, aiming to derive more tradeoffs between the list size and the running time.

3.1 The Single List k-Sum Problem

The previous work in [34] indicates that the LF(k) BKW reduction, in nature, is to find a number of k-tuples from a single list that add to 0 in the last b entries. Currently, this problem is solved by Wagner's k-tree algorithm [33], which is tailored especially to the well known *multi-list* k-sum problem, i.e., each operand comes from a different list. Thus, in the single list setting, a large number of possible sums will be ignored, which has to be compensated by the enlarged size of each list to get a desirable number of solutions. The single-list k-sum problem, introduced in [31], is formally described as follows.

Single list k-sum problem. Given one list L of size m with elements drawn uniformly at random from GF(2)^n, find k distinct vectors $x_1 \in L, \cdots, x_k \in L$ such that $x_1 + \cdots + x_k = 0$.

This single-list problem has a number of natural applications, e.g., the above problem of finding a number of k-tuples from a list that add to 0 on a block of b bits reduces more naturally to the single-list k-sum problem than to the usual multi-list k-sum problem. Other applications include finding parity checks in a (fast) correlation attack, and finding a sparse multiple of a given polynomial also reduces more naturally to the single-list k-sum problem.

As stated above, though dividing the input list L into k distinct sub-lists is feasible in some scenarios, this artificially reduces the effective list size by a factor of k and loses a large number of possible solutions. It is desirable to provide an algorithm to avoid this restriction.

Let $k = 2^p$ be a power of 2. To ensure the existence of at least one solution, the initial list size $m \triangleq m_0$ should satisfy $\binom{m}{k} \geq 2^n$, i.e., $m \geq (k!)^{1/k}2^{n/k}$. In general, if we are expected to have 2^s ($s \geq 0$) solutions, then the initial list size should satisfy

$$m \geq (k!)^{1/k}2^{(n+s)/k} > 2^{\frac{k-1}{k}}2^{\frac{n+s}{k}}. \tag{1}$$

In the following, we will always assume p, m, n and s are given and Eq.(1) holds. Let $\mathrm{low}_l(x)$ be the least significant l bits of x, the algorithm will proceed in p rounds and the general framework is as follows.

In the first round, a new list L_1 is created from the original list L composing of all the sums of $x_1 + x_2$ with $x_1 \neq x_2, x_i \in L$ such that x_1 and x_2 collide on their

first l_1 bits[7]. Thus, all the vectors in L_1 are zero on the first l_1 bits[8], which are sum of the form x_1+x_2 with $x_1 \in L, x_2 \in L, x_2 \neq x_1$. This process can be fulfilled by a sort-and-merge procedure. The m vectors in the initial list L are first sorted into 2^{l_1} equivalence classes according to their values on the first l_1 bits, thus any two vectors in the same equivalence class will have the same value on the specified l_1 positions. Next, let us look at each pair of vectors (x_1, x_2) in each equivalence class to create the new list L_1. Here the integer l_1 is an algorithmic parameter to be determined later for the optimal performance. In short, we say that l_1 bits are eliminated after the first round. It is easy to see that L_1 has an expected size of $m_1 \triangleq \mathrm{E}[|L_1|] = \binom{m}{2} 2^{-l_1} \approx m^2 \cdot 2^{-(l_1+1)}$. Next, in the second round, a new

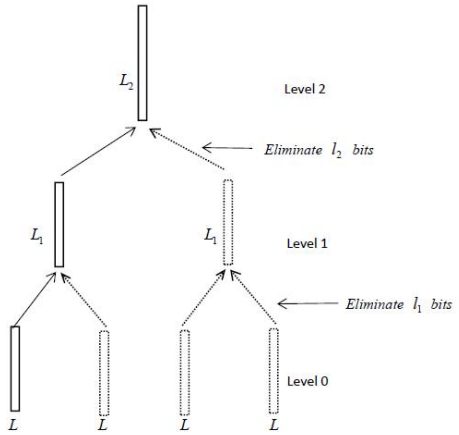

Fig. 1: A pictorial representation of our algorithm for the single-list 4-sum problem

list L_2 is created from L_1 by further eliminating l_2 bits using the same sort-and-merge procedure as that executed in the first round. Now we make the following observation. Let y_1 and y_2 be two different vectors of L_1 in the same equivalence class, then y_1 and y_2 are zero on the first l_1 positions and have the same value on the next l_2 positions, leading to a sum of the form $y_1 + y_2$ which is zero on the first $l_1 + l_2$ bits. Write $y_1 = x_1 + x_2$ ($x_1 \neq x_2$) and $y_2 = x_3 + x_4$ ($x_3 \neq x_4$), the single-list k-sum problem asks to find k *distinct* elements, thus we wish to preserve the sums of $y_1 + y_2$ ($= x_1 + x_2 + x_3 + x_4$) such that x_1, x_2, x_3, x_4 have pairwise different values. In other words, the cases that $x_1 = x_3$ or $x_1 = x_4$ or $x_2 = x_3$ or $x_2 = x_4$ will be ruled out of consideration. Note that the case that $x_1 = x_3$ (resp. $x_1 = x_4$, $x_2 = x_3$, $x_2 = x_4$) will lead to $y_1 + y_2 = x_2 + x_4$ (resp. $x_2 + x_3$, $x_1 + x_4$, $x_1 + x_3$), which is zero on the first $l_1 + l_2$ bits. The expected

[7] The restriction of 0 is not mandatory, only the last l_p-bit value must be 0. This fact makes LF(k) different from the basic LF2 and the proposed hybrid reduction modes meaningful.

[8] Note that for a given l_1, the choice of the least significant bits is not crucial but is introduced here to simplify the presentation. Actually, these l_1 positions can be chosen randomly, so can the following l_2,...,l_p positions.

number of these cases can be estimated as $\binom{m}{2} 2^{-(l_1+l_2)} \approx m^2 \cdot 2^{-(l_1+l_2+1)}$, which is comparatively quite small to the usual cases with non-repeated elements. For the moment, we will do nothing here for these unexpected cases, but add a final sieving process in the last round. Similarly, the expected number of elements in L_2 is $m_2 \triangleq \mathrm{E}[|L_2|] = \binom{m_1}{2} 2^{-l_2} \approx m_1^2 \cdot 2^{-(l_2+1)}$. Note that these repeated samples will not affect the online decoding phase of the LPN solving algorithm, for in such cases, the number of operands is smaller than the normal case, thus the folded noise is also lower than the normal case which is easier to decode in general.

We iterate the above procedure for p rounds and in the i-th round for $i = 1, ..., p$, we create a new list L_i by further eliminating l_i bits, forcing the vectors in L_i to be 0 on the first $\sum_{j=1}^{i} l_j$ bits, which has an expected size of $m_i \triangleq \mathrm{E}[|L_i|] = \binom{m_{i-1}}{2} 2^{-l_i}$. After the p-th round, we obtain a list L_p composing of vectors of the sum of $x_1 + \cdots + x_{k'}$ that are 0 on the first $\sum_{j=1}^{p} l_j$ bits, where $k' \leq k$ is an even number and $x_1, ..., x_{k'}$ are pairwise different vectors from the initial list L. At last, we add a sieving procedure to pick out the solutions that have the repeated indices. We stress here that, in most of the practical applications, the number of solutions found by the algorithm satisfying $k' < k$ is quite small, and it can be ignored compared to the dominant number of the normal cases that $k' = k$. Thus we can make an estimation that the number of solutions found by our algorithm is $m_p \triangleq \mathrm{E}[|L_p|]$. Note that in some cases, we may not need such a sieving process, e.g., when treating with the reduction problem of finding a number of tuples from a list that add to 0 on a block of b bits, we can transform it into the single-list problem and solve it by the above algorithm without the final sieving process. That is, we preserve all the k'-tuples such that $k' \leq k$.

The above process can be visualized as a single-list k-tree algorithm, depicted in Figure 1, where the complete binary tree is of height p, with each node containing a list of vectors. Figure 1 gives a pictorial illustration of the case $k = 4$. The difference between this tree and the famous Wagner's k-tree in [33] is that all the slim boxes in dotted line do *not* really exist, they are just imaginary parts to illustrate the idea of the algorithm. Besides, the list utilized in our tree has a much smaller size compared to that in Wagner's k-tree due to the flexibility in eliminating the different segments.

For the original list L of size m, we write $a_0 \triangleq \log m$, and similarly let $m_i = 2^{a_i}$ be the expected size of the list L_i created in the i-th round. Since $\mathrm{E}[|L_i|] = \binom{\mathrm{E}[|L_{i-1}|]}{2} 2^{-l_i}$, then we have $2^{a_i} = \mathrm{E}[|L_i|] = \binom{E[|L_{i-1}|]}{2} 2^{-l_i} = \frac{2^{a_{i-1}}(2^{a_{i-1}}-1)}{2!} \cdot 2^{-l_i} \approx 2^{2a_{i-1}-(l_i+1)}$, and further we have

$$a_i = 2a_{i-1} - (l_i + 1),\ i = 1, ..., p, \tag{2}$$

where l_i is the number of bits eliminated in the i-th round. As illustrated before, the estimated number of solutions is $m_p = 2^{a_p}$.

The running time of this algorithm is essentially proportional to the size of the lists processed in the algorithm, together with a complexity of the final sieving process, which is usually quite small compared to the whole complexity. Let 2^u be the maximum expected list size that the algorithm has to process, i.e.,

$2^u = \max_{0 \le i \le p-1} 2^{a_i}$, or $u = \max_{0 \le i \le p-1} a_i$, The expected running time of our algorithm is $2^u + 2^s$. Here are two examples.

Example 1. Consider the case that $k = 8$. Let L be a list of size $m = 2^{\frac{3}{4}+\frac{n+s}{4}}$ (we assume $s \leqslant \frac{n}{3} + 1$ and $a_0 = \frac{3}{4} + \frac{n+s}{4}$ is an integer). Following the above algorithm, the choice of $l_1 = l_2 = \frac{n+s-1}{4}$ and $l_3 = \frac{n-s+1}{2}$ will lead to $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = 2^{\frac{3}{4}+\frac{n+s}{4}}$ and $\mathrm{E}[|L_3|] = 2^s$, so we can find an expected number of 2^s solutions[9] in time $3 \cdot 2^{\frac{3}{4}+\frac{n+s}{4}}$ from a list of size $2^{\frac{3}{4}+\frac{n+s}{4}}$. □

Example 2. Consider another case for $k = 8$. Let L be a list of size $m = 2^{\frac{5}{6}+\frac{3n+2s}{12}}$ (we assume that $s \leqslant \frac{3n}{8} + 1$, $a_0 = \frac{5}{6} + \frac{3n+2s}{12}$ is an integer and n is divisible by 4). By choosing $l_1 = \frac{n}{4}$, $l_2 = \frac{n}{4} + \frac{s-1}{3}$ and $l_3 = \frac{n}{2} - \frac{s-1}{3}$ we obtain $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = 2^{\frac{2}{3}+\frac{3n+4s}{12}}$ and $\mathrm{E}[|L_3|] = 2^s$, so we can find an expected number of 2^s solutions in time $2^{\frac{5}{6}+\frac{3n+2s}{12}} + 2 \cdot 2^{\frac{2}{3}+\frac{3n+4s}{12}}$ from a list of size $2^{\frac{5}{6}+\frac{3n+2s}{12}}$. □

3.2 How to Choose the Parameters

The performance of our algorithm is dependent on the parameters l_i that determine the number of bits eliminated in each round. From the above two examples, we can see that the choice of the parameters l_i impact the behavior of the algorithm greatly. For a given instance of the single-list k-sum problem with the specified parameters p, m, n and s satisfying Eq.(1), our goal is to find the optimal choices of the l_is that minimize the running time to find at least 2^s solutions in expectation with a reasonable probability.

Let us start with a discussion about the choice of the l_is with a stronger condition for the list size that

$$m \geq 2^{\frac{\log k}{1+\log k}} 2^{\frac{n+s}{1+\log k}} = 2^{\frac{p}{1+p}} 2^{\frac{n+s}{1+p}}, \text{ or } \log m \geq \frac{p+s+n}{1+p},$$

where $s \leqslant \frac{n}{\log k} + 1$. Under this condition, the single-list k-sum problem can be solved by choosing $l_i = \log m - 1$ for $i = 1, ..., p-1$ and $l_p = 2\log m - s - 1$. In this case, we have $\sum_{i=1}^{p} l_i = (1+p)\log m - (p+s) \geq n$, and $\mathrm{E}[|L_i|] = m$ for $i = 1, ..., p-1$, $\mathrm{E}[|L_p|] = 2^s$, i.e., all the lists except the last one have the same expected size as the initial list. Thus the running time can be estimated as $p \cdot m + 2^s$. In the following, we will always assume that

$$m \leq 2^{\frac{\log k}{1+\log k}} 2^{\frac{n+s}{1+\log k}} = 2^{\frac{p}{1+p}} 2^{\frac{n+s}{1+p}}. \tag{3}$$

As already seen, the key point in our algorithm is to specify an optimal strategy for choosing the number of bits to be eliminated in each round. We will regard this problem as an optimization problem of a linear integer program, which can be solved *analytically*.

The linear program is formulated under the following constraints: first, the maximum expected list size that the algorithm has to process is 2^u, thus we must

[9] There might exist some solutions of 2/4/6-tuples, but this number is quite small and can almost be ignored compared to the total number 2^s.

have $a_i \leq u$ for all $i = 0, 1, ..., p-1$. Second, the last list consists of almost all the solutions found by our algorithm, except for some small number of solutions with repeated indices. Thus, the condition $2^{a_p} \geq s$ should hold, i.e., $a_p \geq s$. Finally, since we want to find sums that are 0 on n bits, we must have $\sum_{i=1}^{p} l_i \geq n$. To sum up, all our needs are translated formally into the following linear integer program.

$$\begin{cases} \textbf{Minimize } u \\ \quad \textbf{s.t.} \quad a_i \leq u,\ i = 0, 1, ..., p-1, \\ \qquad\quad\ a_p \geq s, \\ \qquad\quad\ \sum_{i=1}^{p} l_i \geq n, \\ \qquad\quad\ l_i \geq 0, l_i \text{ are integers},\ i = 1, ..., p \end{cases}$$

Example 3. For $p = 4$, $n = 100$, $m = 2^{17}$ and $s = 2$ for an expected number of four solutions, we obtain an optimal solution of the corresponding integer program, which is $l_1 = 9$, $l_2 = 23$, $l_3 = 23$ and $l_4 = 45$. In this case, we have $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = \mathrm{E}[|L_3|] = 2^{24}, \mathrm{E}[|L_4|] = 4$ and thus the time complexity is $2^{17} + 3 \cdot 2^{24} = 2^{25.6}$. □

Now we try to solve the above integer program analytically. It is worthy noting that comparing with the integer program formulated in [31], here the relationship of the list size in two rounds are different. The following Theorem 1, proved in Appendix A, presents an optimal choice of the l_is for a specified single-list k-sum problem.

Theorem 1. *Given an instance of the single-list k-sum problem with the parameter configuration (p, m, n, s) such that $a_0 = \log m$ is a positive integer, the conditions in (1) and (3) are satisfied, and $s \leq 2\log m - 1$. Define u' as $u' = \frac{n+s+(2^r-1)-2^r a_0}{p-r} + 1$, where r is the least integer such that $n \leq (p-r+1)2^r(a_0 - 1) - s + 1$. Let $u = \lceil u' \rceil$, then we have an optimal solution as follows,*

$$\begin{cases} l_i = 0, & a_i = 2^i a_0 - (2^i - 1),\ \textit{for } 1 \leq i < r \\ l_r = 2^r a_0 - u - (2^r - 1), & a_r = u \\ l_i = u - 1, & a_i = u,\ \textit{for } r < i < p \\ l_p = 2u - s - 1, & a_p = s\ . \end{cases}$$

If we choose the parameters as guided by Theorem 1, then the time complexity of the algorithm is yielded in the following Corollary 1.

Corollary 1. *For the parameters (p, m, n, s) satisfying $(k!)^{1/k} 2^{(n+s)/k} \leq m \leq 2^{\frac{p}{1+p}} 2^{\frac{n+s}{1+p}}$, the single-list k-tree algorithm has an expected running time of $(2^{u^*(p,m,n,s)} + 2^s)$, where $u^*(p, m, n, s)$ is the optimal solution of the integer program formulated and solved in Theorem 1, and the term 2^s represents the complexity of the final sieving procedure for the k distinct elements of $x_1, ..., x_k$.*

3.3 Reduction Schemes of LF(k) for $k = 4, 8, 16$

The LF(k) BKW reduction, introduced in [34], aims to find a number of k-tuples from a single list that add to 0 in the last b entries, where b is a given parameter.

With Wagner's k-tree algorithm [33], the reduction problem of LF(k) for a single solution can be solved in the expected time and space $k2^{b/(1+\lfloor \log k \rfloor)}$ using lists of size $2^{b/(1+\lfloor \log k \rfloor)}$. Moreover, Wagner's algorithm can find $\alpha^{1+\lfloor \log k \rfloor}$ solutions with the α times of the work as for a single solution, as long as $\alpha \leq 2^{b/\lfloor \log k \rfloor (1+\lfloor \log k \rfloor)}$. That is, n_{sol} solutions can be found with both the time and space complexity $\left(k(n_{sol}2^b)^{1/(1+\lfloor \log \kappa \rfloor)}\right)$ from k lists of size $(n_{sol}2^b)^{1/(1+\lfloor \log k \rfloor)}$, as long as $n_{sol} \leq 2^{b/\lfloor \log k \rfloor}$.

In [31], the k-tree algorithm is generalized and the extended k-tree algorithm is proposed to work for any case such that $|L_i| \geq 2^{b/k}$, especially for the case that $2^{b/k} \leq |L_i| \leq 2^{b/(1+\lfloor \log k \rfloor)}$. The crucial difference between the extended k-tree and Wanger's k-tree algorithm lies in the number of bits eliminated in each round, resulting in the difference on the number of solutions and the time/space complexities for finding these solutions from a given number of samples, or equivalently, the difference on the number of samples needed and the time/space complexities for finding a given number of solutions. Specifically, the extended k-tree algorithm provides more tradeoffs between the number of samples needed and the running time. Both the extended k-tree and Wanger's k-tree algorithm are targeted for the multi-list k-sum problem.

As before, suppose there are m samples of n-bit vectors, we wish to find some k-tuples from these samples that add to 0. From the point of multi-list k-sum problem, our first step is to collect these samples into k distinct lists, each of size m/k, and then solve it either by the Wagner's or the extended k-tree algorithm. Take $k = 4$ as an illustrative example. We can find 2^s solutions from $m = 4 \cdot 2^{(n+s)/3}$ samples with the time complexity $T = 6 \cdot 2^{(n+s)/3}$ and space complexity $S = 3 \cdot 2^{(n+s)/3}$, if we choose $l_1 = \frac{n+s}{3}$ and $l_2 = \frac{2n-s}{3}$, due to Wagner's 4-tree algorithm. From the viewpoint of single-list k-sum problem, however, what we need to do first is to collect the samples in one single list of size m, and then solve it by our new algorithm. According to Theorem 1, we can find 2^s solutions from $m' = 2^{(n+s+2)/3}$ ($< m$) samples with the time $T' = 2 \cdot 2^{(n+s+2)/3}$ ($< T$) and space complexity $S' = 2 \cdot 2^{(n+s+2)/3}$ ($< S$) if we choose $l_1 = \frac{n+s-1}{3}$ and $l_2 = \frac{2n-s+1}{3}$. That is, $m' < m$, $T' < T$ and $S' < S$ will hold *simultaneously*. In fact, we have observed that it exhibits better effect on both the list size and the running time by connecting LF(k) with our single-list k-sum problem, than that with the multi-list k-sum problem, not just for this example. Based on this observation, we will present some LF(k) schemes derived from Theorem 1, for $k = 4$, $k = 8$ and $k = 16$, respectively, aiming to derive more tradeoffs between the list size and the running time.

(1). Scheme for LF(4).

Now we wish to find some 4-tuples from the m given samples of n-bit vectors that add to 0. We present one scheme for LF(4) from the aspect of the multi-list and single-list 4-sum problem, respectively, shown below. The details are listed in Table 2.

- For the multi-list 4-sum problem, we have an expected number of 2^s solutions from $m = 4 \cdot 2^{n/3+s/4}$ samples with the time complexity $T = (4 \cdot 2^{n/3+s/4} + 2 \cdot$

$2^{n/3+s/2}$) and space complexity $S = (2^{n/3+s/4} + 2 \cdot 2^{n/3+s/2})$ when choosing $l_1 = \frac{n}{3}$ and $l_2 = \frac{2n}{3}$, due to the extended 4-tree algorithm. In this case, we have $\mathrm{E}[|L_1|] = 2^{n/3+s/2}$ and $\mathrm{E}[|L_2|] = 2^s$.
- For the single-list 4-sum problem, we have an expected number of 2^s solutions from $m' = 2^{3/4+n/3+s/4}$ ($< m$) samples with the time complexity $T' = (2^{3/4+n/3+s/4} + 2^{1/2+n/3+s/2})$ ($< T$) and space complexity $S' = (2^{3/4+n/3+s/4} + 2^{1/2+n/3+s/2})$ ($< M$) if we choose $l_1 = \frac{n}{3}$ and $l_2 = \frac{2n}{3}$, according to Theorem 1. In this case, we have $\mathrm{E}[|L_1|] = 2^{1/2+n/3+s/2}$ and $\mathrm{E}[|L_2|] = 2^s$.

Example 4. Suppose now there are 2^{29} samples of 60-bit vectors available, we wish to find 2^{33} 4-tuples from these samples that add to 0. It can fit into the single-list k-tree algorithm with the following parameters $k = 4$, $m = 2^{29}$, $n = 60$ and $s = 33$. Theorem 1 dictates that we should choose $l_1 = \frac{n}{3} = 20$ and $l_2 = \frac{2n}{3} = 40$, resulting in $\mathrm{E}[|L_1|] = 2^{37}$ and $\mathrm{E}[|L_2|] = 2^{33}$. Thus we can find an expected number of 2^{33} solutions[10] in time $2^{29} + 2^{37} = 2^{37.0}$ from a list of size $m = 2^{29}$. In contrast, from the viewpoint of multi-list 4-sum problem, we need to put these samples into 4 distinct lists, each of size $2^{29}/4 = 2^{27}$. Due to the extended 4-tree algorithm, we should choose $l_1 = 15$ and $l_2 = 45$, leading to $\mathrm{E}[|L_1|] = 2^{39}$ and $\mathrm{E}[|L_2|] = 2^{33}$. Thus we can find an expected number of 2^{33} solutions in time $4 \cdot 2^{27} + 2 \cdot 2^{39} = 2^{40.0}$ from 4 lists, each of size 2^{27}. □

Table 2: A tradeoff point of LF(4) for finding 2^s solutions

Type	l_1	l_2	*Number of samples* ($\log_2 m$)	*Time*	*Space*
multi-list	$\frac{n}{3}$	$\frac{2n}{3}$	$2 + \frac{n}{3} + \frac{s}{4}$	$2^{1+n/3+s/2}$	$2^{1+n/3+s/2}$
single-list	$\frac{n}{3}$	$\frac{2n}{3}$	$\frac{3}{4} + \frac{n}{3} + \frac{s}{4}$	$2^{1/2+n/3+s/2}$	$2^{1/2+n/3+s/2}$

(2). Scheme for LF(8).

Suppose now there are m samples of n-bit vectors available, we wish to find some 8-tuples from these samples that add to 0. Similar to the LF(4) case, we present a scheme for LF(8), shown below. The details are listed in Table 3.

- For the multi-list 8-sum problem, we have an expected number of 2^s solutions from $m = 8 \cdot 2^{n/4+s/6}$ samples with the time $T = 8 \cdot 2^{n/4+s/6} + 6 \cdot 2^{n/4+s/3}$ and space complexity $S = 2^{n/4+s/6} + 3 \cdot 2^{n/4+s/3}$ when choosing $l_1 = \frac{n}{4}$, $l_2 = \frac{n}{4} + \frac{s}{3}$ and $l_3 = \frac{n}{2} - \frac{s}{3}$, due to the extended 8-tree algorithm. In this case, we have $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = 2^{n/4+s/3}$ and $\mathrm{E}[|L_3|] = 2^s$.
- For the single-list 8-sum problem, we have an expected number of 2^s solutions from $m' = 2^{5/6+n/4+s/6}$ samples with the time $T' = \tilde{O}(2^{5/6+n/4+s/6} + 2 \cdot 2^{2/3+n/4+s/3})$ and space complexity $S' = 2^{5/6+n/4+s/6} + 2^{2/3+n/4+s/3}$ when choosing $l_1 = \frac{n}{4}$, $l_2 = \frac{n}{4} + \frac{s-1}{3}$ and $l_3 = \frac{n}{2} - \frac{s-1}{3}$, according to Theorem 1. In this case, we have $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = 2^{2/3+n/4+s/3}$ and $\mathrm{E}[|L_3|] = 2^s$.

[10] There might exist some solutions of 2-tuples, the number can be computed as $\binom{m}{2} 2^{-(l_1+l_2)} \approx 0$.

Example 5. Suppose now we have 2^{20} samples of 60-bit vectors and wish to find 2^{25} 8-tuples from these samples that add to 0. It can fit into the single-list k-tree algorithm with the parameters $k = 8$, $m = 2^{20}$, $n = 60$ and $s = 25$. Theorem 1 suggests that we should choose $l_1 = \frac{n}{4} = 15$, $l_2 = \frac{n}{4} + \frac{s-1}{3} = 23$ and $l_3 = \frac{n}{2} - \frac{s-1}{3} = 22$, resulting in $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = 2^{24}$ and $\mathrm{E}[|L_3|] = 2^{25}$. Thus we can find an expected number of 2^{25} solutions[11] in time $2^{20} + 2 \cdot 2^{24} = 2^{25.0}$ from a list of size $m = 2^{20}$. In contrast, from the viewpoint of the multi-list k-sum problem, we need to throw these samples into 8 distinct lists, each of size $2^{20}/8 = 2^{17}$. Due to the extended 8-tree algorithm, we should choose $l_1 = 8$, $l_2 = 26$ and $l_3 = 27$, leading to $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = 2^{26}$ and $\mathrm{E}[|L_3|] = 2^{25}$. Thus we can find an expected number of 2^{25} solutions in time $8 \cdot 2^{17} + 4 \cdot 2^{26} + 2 \cdot 2^{25} = 2^{28.3}$ from 8 lists, each of size 2^{17}. □

Table 3: A tradeoff point of LF(8) for finding 2^s solutions

Type	l_1	l_2	l_3	*Number of samples* ($\log_2 m$)	*Time*	*Space*
multi-list	$\frac{n}{4}$	$\frac{n}{4} + \frac{s}{3}$	$\frac{n}{2} - \frac{s}{3}$	$3 + \frac{n}{4} + \frac{s}{6}$	$6 \cdot 2^{\frac{n}{4}+\frac{s}{3}}$	$3 \cdot 2^{\frac{n}{4}+\frac{s}{3}}$
single-list	$\frac{n}{4}$	$\frac{n}{4} + \frac{s-1}{3}$	$\frac{n}{2} - \frac{s-1}{3}$	$\frac{5}{6} + \frac{n}{4} + \frac{s}{6}$	$2 \cdot 2^{\frac{2}{3}+\frac{n}{4}+\frac{s}{3}}$	$2^{\frac{2}{3}+\frac{n}{4}+\frac{s}{3}}$

(3). Scheme for LF(16).

Suppose now we have m samples of n-bit vectors available, we wish to find some 16-tuples from these samples that add to 0. Similarly, we present one scheme for LF(16), shown as follows. The details are listed in Table 4.

- For the multi-list 16-sum problem, we have an expected number of 2^s solutions from $m = 16 \cdot 2^{3n/16+s/8}$ samples with the time complexity $T = 16 \cdot 2^{3n/16+s/8} + 14 \cdot 2^{5n/24+s/4}$ and the space complexity $M = 2^{3n/16+s/8} + 4 \cdot 2^{5n/24+s/4}$. when choosing $l_1 = \frac{n}{6}$, $l_2 = l_3 = \frac{5n}{24} + \frac{s}{4}$ and $l_4 = \frac{5n}{12} - \frac{s}{2}$, due to the extended 8-tree algorithm. In this case, we have $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = \mathrm{E}[|L_3|] = 2^{5n/24+s/4}$ and $\mathrm{E}[|L_4|] = 2^s$.
- For the single-list 16-sum problem, we have an expected number of 2^s solutions from $m' = 2^{7/8+3n/16+s/8}$ samples with the time and space complexity $T' = (2^{7/8+3n/16+s/8} + 3 \cdot 2^{3/4+5n/24+s/4})$ and space complexity $Comp' = (2^{7/8+3n/16+s/8} + 2^{3/4+5n/24+s/4})$ when choosing $l_1 = \frac{n}{6}$, $l_2 = l_3 = \frac{5n}{24} + \frac{s-1}{4}$ and $l_4 = \frac{5n}{12} - \frac{s-1}{2}$, according to Theorem 1. In this case, we have $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = \mathrm{E}[|L_3|] = 2^{3/4+5n/24+s/4}$ and $\mathrm{E}[|L_4|] = 2^s$.

Example 6. Suppose now there are 2^{15} samples of 60-bit vectors available, we wish to find 2^{23} 16-tuples from these samples that add to 0. It can fit into the single-list k-tree algorithm with the following parameters $k = 16$, $m = 2^{15}$, $n = 60$ and $s = 23$. From Theorem 1, we choose $l_1 = \frac{n}{6} = 10$, $l_2 = l_3 = \frac{5n}{24} + \frac{s-1}{4} = 18$ and $l_4 = \frac{5n}{12} - \frac{s-1}{2} = 14$, resulting in $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = \mathrm{E}[|L_3|] = 2^{19}$ and

[11] There might exist some solutions of 2/4/6-tuples, though this number is quite small. We will preserve these solutions since they will generate the bias benefit after the reduction procedure.

$\mathrm{E}[|L_4|] = 2^{23}$. Thus we can find an expected number of 2^{23} solutions[12] in time $2^{15} + 3 \cdot 2^{19} = 2^{20.6}$ from a list of size $m = 2^{15}$. On the other side, from the viewpoint of the multi-list k-sum problem, we need to put these samples into 16 lists, each of size $2^{15}/16 = 2^{11}$. Due to the extended 4-tree algorithm, we should choose $l_1 = 1$, $l_2 = l_3 = 21$ and $l_4 = 19$, leading to $\mathrm{E}[|L_1|] = \mathrm{E}[|L_2|] = \mathrm{E}[|L_3|] = 2^{21}$ and $\mathrm{E}[|L_4|] = 2^{23}$. Thus we can find an expected number of 2^{23} solutions in time $16 \cdot 2^{11} + 8 \cdot 2^{21} + 4 \cdot 2^{21} + 2 \cdot 2^{21} = 2^{24.8}$ from 16 lists, each of size 2^{11}. □

Table 4: A tradeoff point of LF(16) for finding 2^s solutions

Type	l_1	$l_2 (= l_3)$	l_4	*Number of samples* $(\log_2)$	*Time*	*Space*
multi-list	$\frac{n}{6}$	$\frac{5n}{24} + \frac{s}{4}$	$\frac{5n}{12} - \frac{s}{2}$	$4 + \frac{3n}{16} + \frac{s}{8}$	$14 \cdot 2^{\frac{5n}{24} + \frac{s}{4}}$	$4 \cdot 2^{\frac{5n}{24} + \frac{s}{4}}$
single-list	$\frac{n}{6}$	$\frac{5n}{24} + \frac{s-1}{4}$	$\frac{5n}{12} - \frac{s-1}{2}$	$\frac{7}{8} + \frac{3n}{16} + \frac{s}{8}$	$3 \cdot 2^{\frac{3}{4} + \frac{5n}{24} + \frac{s}{4}}$	$2^{\frac{3}{4} + \frac{5n}{24} + \frac{s}{4}}$

4 The Single List k-sum Problem over Large Alphabets

Taking into account the fact that there are more and more cryptographic primitives working on some larger alphabets, we study the single list k-sum problem over some non-binary alphabets, described as follows.

Let r be a prime and $R = r^w$ be a power of r. We are given one list L of size m with elements drawn randomly from $\mathrm{GF}(R)^n$, we wish to find $x_1 \in L, ..., x_k \in L$ and $\lambda_1, ..., \lambda_k \in \mathrm{GF}(R)^*$ such that $\lambda_1 x_1 + \cdots \lambda_k x_k = 0$.

When solving LWE, the reduction problem of finding a number of k-tuples from a list that add to 0 on a block of b entries reduces naturally to the single-list k-sum problem over some finite field. In stream ciphers, it can be used to find parity checks in a (fast) correlation attack over some extension field.

In the following, we develop an alternative version of the single list k-tree algorithm tailored to the non-binary case. Let us start with a discussion about the merging procedure. Our goal is to find a more involved merging procedure which ensures that only a single scalar multiple of each relevant linear combination is retained in the list at each round of the algorithm. Given one list L of size m and an integer l representing the number of positions to eliminate, we wish to create a merged list L' from L composing of all the sums of $\lambda_1 x_1 + \lambda_2 x_2$ with $x_1 \in L, x_2 (\neq x_1) \in L$ and $\lambda_1, \lambda_2 \in \mathrm{GF}(R)^*$ such that it is zero on the first l positions. It is required that the merged list L' should have similar properties as that in [31], i.e., validity, completeness and minimality. To achieve this, we proceed as follows. First, each vector in L is transformed into the normalized form of $(*, ..., *, 1, 0, ..., 0)$ by multiplying some suitable constant. Next, m vectors in L are sorted in equivalence classes according to their values on their

[12] There might exist some solutions of 2/4/6/8/10/12/14-tuples, though this number is quite small. Similarly as before, we will preserve these solutions since they will generate the bias benefited after the reduction procedure.

least significant l positions. Let x_1, x_2 be two different normalized vectors of the equivalence class in which vectors are zero on the least significant l positions, then there are exactly $R-1$ distinct valid sums of x_1 and x_2 of the form $x_1 + \lambda x_2$ for all $\lambda \in \mathrm{GF}(R)^*$, such that it is zero on the least significant l positions. Let y_1, y_2 be two different vectors in other equivalence classes, then y_1 and y_1 have the same value (not zero) on the l positions, and the only possible valid sum of $\lambda_1 x_1 + \lambda_2 x_2$ such that it is zero on the least significant l positions must satisfy $\lambda_1 + \lambda_2 = 0, \lambda_i \in \mathrm{GF}(R)^*$, i.e., we still have $R-1$ distinct valid sums of the form $\lambda x_1 + \lambda x_2$ for all $\lambda \in \mathrm{GF}(R)^*$. Finally, the merged list L' is created with running time m. By the birthday paradox, L' has an expected size of $m' = \binom{m}{2}(R-1)R^{-l} = \frac{m^2}{2}(R-1)R^{-l}$.

Let $k = 2^p$ be a power of 2. Similar to the k-tree algorithm over GF(2) in Section 3, this algorithm will still process in p rounds. Let l_i be the number of positions to be eliminated in each round, we will formalize a linear program with the goal of finding the optimal choices of the l_is for a specified single-list k-sum problem over $\mathrm{GF}(R)$, where $R = r^w$. Define $b_i = \log_R m_i$, where m_i is the expected number of size created in the ith round for $i = 1, ..., p$ and $m_0 = m$. From the above, we have $m_i = \binom{m_{i-1}}{2}(R-1)R^{-l_i} = \frac{m_{i-1}^2}{2}(R-1)R^{-l_i}$ for $i = 1, ..., p$, or equivalently,

$$\begin{cases} b_0 = \log_R m \\ b_i = 2b_{i-1} - l_i + \delta_R, \ i = 1, ..., p \end{cases}$$

where $\delta_R = \log_R \frac{R-1}{2}$ is a constant. Suppose now we specify that the expected number of solutions found by the algorithm should be at least R^s, this will lead to the following linear integer program:

$$\begin{cases} \textbf{Minimize} \ \ u \\ \quad \textbf{s.t.} \quad b_i \le u, \ i = 0, 1, ..., p-1, \\ \qquad\quad b_p \ge s, \\ \qquad\quad \sum_{i=1}^{p} l_i \ge n, \\ \qquad\quad l_i \ge 0, l_i \text{ are integers}, \ i = 1, ..., p. \end{cases}$$

Similar to the range bound of m given in the binary case by conditions (1) and (3), we have $\frac{(k!)^{\frac{1}{k}} R^{\frac{n+s}{k}}}{(R-1)^{\frac{1}{k}}} \le m \le \frac{2^{\frac{p}{1+p}} R^{\frac{n+s}{1+p}}}{(R-1)^{\frac{p}{1+p}}}$. The lower bound for m is determined by the existence of the R^s solutions with a reasonable probability, while the upper bound corresponds to the size where the following style algorithm can be implemented. Specifically, let us look at the choice of l_i with a stronger condition for the list size m that $m \ge \frac{2^{\frac{p}{1+p}} R^{\frac{n+s}{1+p}}}{(R-1)^{\frac{p}{1+p}}}$, where $s \leqslant \frac{n}{\log_R \kappa} - \delta_R$. Under this condition, the single-list k-sum problem over $\mathrm{GF}(R)$ can be solved by choosing $l_i = \log m + \delta_R$ for $i = 1, ..., p-1$ and $l_p = 2\log m - s + \delta_R$. In this case, we have $\sum_{i=1}^{p} l_i = (1+p)\log m - s + p\delta_R \ge n$, and $\mathrm{E}[|L_i|] = m$ for $i = 1, ..., p-1$, $\mathrm{E}[|L_p|] = R^s$, i.e., all the lists except the last one have the same expected size as the initial list. Thus the running time can be estimated as $p \cdot m + R^s$, where the

term R^s represents the complexity of the final sieving process for the k distinct elements of $x_1, ..., x_k$. In the following, we will always assume

$$\frac{(k!)^{\frac{1}{k}} R^{\frac{n+s}{k}}}{(R-1)^{1-\frac{1}{k}}} \leq m \leq \frac{2^{\frac{p}{1+p}} R^{\frac{n+s}{1+p}}}{(R-1)^{\frac{p}{1+p}}}. \tag{4}$$

Correspondingly, we have the following result on the optimal choice of l_i for $i = 1, ..., p$ (we still first relax the constraint that the l_i should be integer). The proof of the following Theorem 2 is analogous to the proof of Theorem 1, so we will not go into the details here, and only illustrate it by a simple example.

Theorem 2. *Given an instance of the single-list k-sum problem over $GF(R)$ with the parameters (p, m, n, s) such that the condition (4) holds and $s \leq 2 \log m + \delta_R$. Define u' as $u = \frac{n+s-(2^r-1)\delta_R - 2^r b_0}{p-r} - \delta_R$, where r is the least integer such that $n \leq (p-r+1)2^r(b_0 + \delta_R) - s - \delta_R$, then we have an optimal solution of the above linear program as follows,*

$$\begin{cases} l_i = 0, & b_i = 2^i b_0 + (2^i - 1)\delta_R, \text{ for } 1 \leq i < r \\ l_r = 2^r b_0 - u + (2^r - 1)\delta_R, & b_r = u \\ l_i = u + \delta_R, & b_i = u, \text{ for } r < i < p \\ l_p = 2u - s + \delta_R, & b_p = s \end{cases}$$

Example 7. Suppose we work over $\mathrm{GF}(2^6)$ with the parameters $k = 8$ and $n = 18$, we compute the range of m by inequality (4) as $2^{10.18} \leq m \leq 2^{23.27}$. Now we set $s = 0$ and the size of the input list $m = 2^{22}$, according to Theorem 2, we obtain a maximal list length of $R^u \approx 2^{25}$. □

5 Hybrid BKW Reduction Modes

Recall that in the collision procedure of the LPN solving algorithm, the size of the secret is further reduced by xoring k-tuples of vectors that add to 0 in the last b entries. In [34], as shown in Sect. 2, the reduction mode of LF1, LF2 and LF(4) are adopted in this step, respectively.

In this section, we propose more reduction modes to employ different kinds of reduction mode together, called hybrid reduction modes, i.e., LF(k) + LF2, aiming to find some better complexity tradeoffs. Especially we present the details of LF(4) + LF2, LF(8) + LF2 and LF(16) + LF2, which are based on the scheme of LF(k), $k = 4, 8, 16$, as illustrated in Sect. 3.

It is worthy noting that with carefully chosen parameters, the overall complexity is not increased; while at the same time, new and better tradeoffs have been achieved.

For the hybrid BKW reduction mode LF(k)+LF2, we will iteratively process t steps of the reduction mode LF(k), followed by t' steps of LF2. Denote $n[i]$, $i = 1, 2, ..., t+t'$, as the expected number of samples via the i-th BKW reduction step, where $n[0] = n$, $n[t + t'] = m$ and m is the number of queries required for the final solving phase.

For the reduction of LF(k), we first need to find sufficient k-tuples of columns that add to 0 in some bits, and then calculate the sum of each k-tuple. For $i = 1, ..., t$, we need to find $n[i]$ such k-tuples from the $n[i-1]$ samples iteratively. According to Theorem 1, we can derive various choices for both the relationship of $n[i]$ and $n[i-1]$ and the running time for finding these solutions, by connecting LF(k) with the single-list k-sum problem. Let b_i be the number of bits eliminated at the i-th reduction step and $B_i = \sum_{j=1}^{i} b_j$. According to the scheme of LF(k), $k = 4, 8, 16$, in Sect. 3.3, we have

- For LF(4), we have $n[i-1] = 2^{3/4} n[i]^{1/4} 2^{b_i/3}$ and the cost of finding $n[i]$ solutions is $(2^{3/4} n[i]^{1/4} + 2^{1/2} n[i]^{1/2}) 2^{b_i/3}$ both in time and space.
- For LF(8), we have $n[i-1] = 2^{5/6} n[i]^{1/6} 2^{b_i/4}$ and the cost of finding $n[i]$ solutions is $(2^{5/6} n[i]^{1/6} + 2 \cdot 2^{2/3} n[i]^{1/3}) 2^{b_i/4}$ in time and $(2^{5/6} n[i]^{1/6} + 2^{2/3} n[i]^{1/3}) 2^{b_i/4}$ in space.
- For LF(16), we have $n[i-1] = 2^{7/8} n[i]^{1/8} 2^{3b_i/16}$ and the cost of finding $n[i]$ solutions is $(2^{7/8} n[i]^{1/8} 2^{3b_i/16} + 3 \cdot 2^{3/4} n[i]^{1/4} 2^{5b_i/24})$ in time and $(2^{7/8} n[i]^{1/8} 2^{3b_i/16} + 2^{3/4} n[i]^{1/4} 2^{5b_i/24})$ in space.

For the merging procedure of LF(k), just as stated in [34], we divide each k-tuple of $(k+1-B_i)$-bit columns into $\lceil \frac{k+1-B_i}{e} \rceil$ parts, and read the sum of each part directly from the table storing all the possible additions of k e-bit vectors, where e is a properly chosen integer. Thus the cost of the merging procedure is $\lceil \frac{k+1-B_i}{e} \rceil n[i]$, and the table can be constructed with a time complexity of $PC_{21} = \sum_{i=2}^{k} e 2^{e-1} \binom{2^e-2}{i-1}$ and a memory complexity of $e2^{ke}$ to store the table. For $k = 4, 8, 16$, the time/memory complexities for the t steps of LF(k) are as follows:

- For LF(4), $C_{21} = \sum_{i=1}^{t} \left(\lceil \frac{k+1-B_i}{e} \rceil n[i] + (2^{3/4} n[i]^{1/4} + 2^{1/2} n[i]^{1/2}) 2^{b_i/3} \right)$ and $M_{21} = e2^{4e} + (2^{3/4} n[i]^{1/4} + 2^{1/2} n[i]^{1/2}) 2^{b_i/3}$.
- For LF(8), $C_{21} = \sum_{i=1}^{t} \left(\lceil \frac{k+1-B_i}{e} \rceil n[i] + (2^{5/6} n[i]^{1/6} + 2 \cdot 2^{2/3} n[i]^{1/3}) 2^{b_i/4} \right)$ and $M_{21} = e2^{8e} + (2^{5/6} n[i]^{1/6} + 2^{2/3} n[i]^{1/3}) 2^{b_i/4}$.
- For LF(16), we have $C_{21} = \sum_{i=1}^{t} \left(\lceil \frac{k+1-B_i}{e} \rceil n[i] + (2^{7/8} n[i]^{1/8} 2^{3b_i/16} + 3 \cdot 2^{3/4} n[i]^{1/4} 2^{5b_i/24}) \right)$ and $M_{21} = e2^{16e} + (2^{7/8} n[i]^{1/8} 2^{3b_i/16} + 2^{3/4} n[i]^{1/4} 2^{5b_i/24})$.

After t steps of LF(k), we further process t' steps of LF2. As shown in [34], $n[i] = \binom{n[i-1]}{2} 2^{-b_i}$, i.e., $n[i-1] \approx (2!)^{1/2} n[i]^{1/2} 2^{b_i/2}$, $i = t+1, ..., t+t'$. Similarly, t' steps LF2 will take a cost of $C_{22} = \sum_{i=1}^{t'} \left(\left\lceil \frac{k+1-B_{t+i}}{f} \right\rceil n[t+i] + n[t+i-1] \right)$, and also a pre-computation of $PC_{22} = f2^{f-1}(2^f - 2)$ in time and $M_{22} = f2^{2f}$ in memory, where f is a properly chosen integer.

To sum up, the time and memory complexity of the LF(k) + LF2 reduction mode for $k = 4, 8, 16$ are $C_2 = C_{21} + C_{22}$ and $M_2 = M_{21} + M_{22}$, respectively, and also a pre-computation of $PC_2 = PC_{21} + PC_{22}$ is needed. Note that the bias of the noise is sharply reduced from the original bias $\varepsilon (= 1 - \eta)$ to $\varepsilon^{k^t 2^{t'}}$ after this kind of collision procedure.

6 Improving the Gaussian Elimination Step

It is observed in [9] that the previous optimized method in [34] in the Gaussian elimination step still takes a rather huge complexity due to some very large table pre-computed and read frequently in the attack. Actually, the dominant part in the Gaussian elimination step is to compute the matrix multiplication **DG** in Section 2.

Now we present some further improvements on this procedure to clarify the controversial issues, which are described as follows. Note that here we take the random access machine model [12] which has an unit-cost for read/write access to all of its memory and is adopted by almost all the cryptanalysis literatures by default. First, a tables storing $[\boldsymbol{x}, \mathbf{D}_i\boldsymbol{x}^T]_{\boldsymbol{x}\in \mathrm{GF}(2)^s}$ are constructed in [34], and

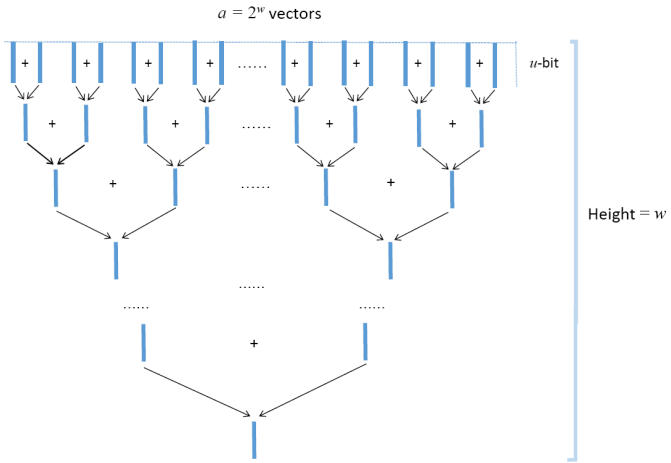

Fig. 2: Schematic of addition of $a = 2^w$ u-bit vectors

are stored with a memory complexity of $2^s(s+k)a$. Here we can optimize the required memory slightly by storing only $[\mathbf{D}_i\boldsymbol{x}^T]_{\boldsymbol{x}\in \mathrm{GF}(2)^s}$ in the table indexed by $i = 1, ..., a$, where each row is indexed by $\boldsymbol{x} \in \mathrm{GF}(2)^s$, thus the memory is reduced from $2^s(s+k)a$ to $M_{11} = 2^s ka$. Second, recall that the addition of a k-bit vectors via a table look-up is executed as follows in [34]. First construct a table storing all the possible additions of a u-bit vectors and read it $d = \lceil \frac{k}{u} \rceil$ times to obtain the sum of a k-bit vectors, where u is a properly chosen integer. Let $T_+(a, k)$ (resp. $T_+(a, u)$) be the cost of computing the addition of a k-bit (resp. u-bit) vectors, then $T_+(a, k) = d \cdot T_+(a, u)$. Based on this, let us do something different. Instead of building one large table storing all the possible additions of a u-bit vectors, we consider to construct a relatively small table. Precisely, we first choose an integer a such that a is the power of some integer q, i.e., $a = q^w$ for some integer w. Then for a properly chosen integer u, we construct a small table storing all the possible additions of q u-bit vectors. Thus the addition of a u-bit

vectors can be obtained by reading this small table $1+q^1+...+q^{w-1}=\frac{q^w-1}{q-1}$ times, depicted in Figure 2 and in Figure 3 in Appendix B. Denote $T_+(q,u)$ as the cost of calculating the addition of q u-bit vectors, we obviously have $T_+(a,u)=\frac{q^w-1}{q-1}\cdot T_+(q,u)$, and further $T_+(a,k)=\frac{q^w-1}{q-1}d\cdot T_+(q,u)$. That is, the addition of a k-bit vectors can finally be derived by reading this table $\frac{q^w-1}{q-1}\cdot d$ times. Thus the time complexity of the Gaussian elimination step is changed from $(n-k)(a+d)$ to $C_1=(n-k)\left(a+\frac{q^w-1}{q-1}d\right)$, and the complexity for constructing the table storing all the possible additions of q u-bit vectors is changed from $\sum_{i=2}^{a}u2^{u-1}\binom{2^u-2}{i-1}$ to $PC_{12}=\sum_{i=2}^{q}u2^{u-1}\binom{2^u-2}{i-1}$ in time and the memory[13] needed is changed from $\sum_{i=2}^{a}\frac{i+1}{i}u2^{u-1}\binom{2^u-2}{i-1}$ to $M_{12}=u2^{qu}$.

Note that the memory complexity has been significantly reduced especially for $q=2$ or $q=3$ and $w>1$. Hence, with carefully chosen parameters a and u, we have a good chance to derive better tradeoffs between the complexities of C_1, PC_{11}, M_{11}, PC_{12} and M_{12}. Here is an illustrative example.

Example 8. Consider the case that $a=2^3$ ($q=2$, $w=3$) and $u=32$. To calculate the addition of 8 32-bit vectors such as $\boldsymbol{u}\triangleq\boldsymbol{u}_1+\boldsymbol{u}_2+\boldsymbol{u}_3+\boldsymbol{u}_4+\boldsymbol{u}_5+\boldsymbol{u}_6+\boldsymbol{u}_7+\boldsymbol{u}_8$, we first construct a table storing all the possible additions of two 32-bit vectors, i.e., $[\boldsymbol{v}_1+\boldsymbol{v}_2]_{\boldsymbol{v}_i\in\mathrm{GF}(2)^{32}}$, where each row is indexed by $(\boldsymbol{v}_1,\boldsymbol{v}_2)$. Then we read the table in the following order. First, read the rows indexed by $(\boldsymbol{u}_1,\boldsymbol{u}_2)$, $(\boldsymbol{u}_3,\boldsymbol{u}_4)$, $(\boldsymbol{u}_5,\boldsymbol{u}_6)$ and $(\boldsymbol{u}_7,\boldsymbol{u}_8)$ to derive $\boldsymbol{u}_{12}\triangleq\boldsymbol{u}_1+\boldsymbol{u}_2$, $\boldsymbol{u}_{34}\triangleq\boldsymbol{u}_3+\boldsymbol{u}_4$, $\boldsymbol{u}_{56}\triangleq\boldsymbol{u}_5+\boldsymbol{u}_6$ and $\boldsymbol{u}_{78}\triangleq\boldsymbol{u}_7+\boldsymbol{u}_8$, respectively. Second, read the rows indexed by $(\boldsymbol{u}_{12},\boldsymbol{u}_{34})$, $(\boldsymbol{u}_{56},\boldsymbol{u}_{78})$ to derive $\boldsymbol{u}_{1234}\triangleq\boldsymbol{u}_{12}+\boldsymbol{u}_{34}$, $\boldsymbol{u}_{5678}\triangleq\boldsymbol{u}_{56}+\boldsymbol{u}_{78}$, respectively. Finally, read the rows indexed by $(\boldsymbol{u}_{1234},\boldsymbol{u}_{5678})$ to derive $\boldsymbol{u}_{1234}+\boldsymbol{u}_{5678}$, which is the final result of $\boldsymbol{u}$. In conclusion, we obtain the addition of 8 32-bit vectors by reading the table $4+2+1=7$ times. □

Note that in Fig.2, the table look-ups at each level of the tree could be paralleled to reduce the time complexity to that of only one table look-up. Thus for the whole tree, the parallel time complexity is w table look-ups[14]. We emphasize that all the above techniques could be further exploited in some other applications [1, 11] on Gaussian elimination.

7 The Integrated Framework

In this section, we will present the complexity issues of the improved algorithm, in which the newly proposed techniques are integrated into the LPN solving framework. It is natural to derive a refined framework for solving LPN by integrating the single-list k-sum algorithm, hybrid BKW reduction mode and the improved Gaussian elimination method into the previous one, i.e., we could

[13] In [34], only instances of the sum problem with non-repeating and non-zero terms are stored in the table, but this has been questioned in [9]. Here we clarify the controversial issue by storing in the table all the 2^{qu} instances.

[14] For the complexity results in the paper, we did not take the parallel technique so far, though it would result in even better results.

take the best possible tradeoff in each step (**Step 0** to **Step 5**) to get better attacks on generic LPN. The complexity issues of the new framework is as follows. The final bias of the noise is $\varepsilon_f = \varepsilon_{cp}\varepsilon_{cc}$, where $\varepsilon_{cp} = \varepsilon^{4^t 2^{t'}}$ is for the

Table 5: The optimal cascaded perfect codes chosen for solving the $(512, \frac{1}{8})$, $(532, \frac{1}{8})$, $(592, \frac{1}{8})$-LPN instances

(k, η)-LPN	$[k_2, l]$	Cascaded perfect codes	Number of chunks	$\log_2 \varepsilon_{cc}$
$(512, \frac{1}{8})$	[172,61]	$1 \cdot \mathcal{H}_1 + 6 \cdot \mathcal{R}_2 + 7 \cdot \mathcal{R}_3 + 4 \cdot \mathcal{G}$	18	-15.3801
	[174,62]	$8 \cdot \mathcal{R}_2 + 6 \cdot \mathcal{R}_3 + 4 \cdot \mathcal{G}$	18	-15.4742
	[177,62]	$1 \cdot \mathcal{H}_1 + 7 \cdot \mathcal{R}_2 + 7 \cdot \mathcal{R}_3 + 4 \cdot \mathcal{G}$	19	-16.0050
$(532, \frac{1}{8})$	[181,63]	$8 \cdot \mathcal{R}_2 + 7 \cdot \mathcal{R}_3 + 4 \cdot \mathcal{G}$	19	-16.4374
	[182,64]	$11 \cdot \mathcal{R}_2 + 5 \cdot \mathcal{R}_3 + 4 \cdot \mathcal{G}$	20	-16.3860
	[185,64]	$1 \cdot \mathcal{H}_1 + 10 \cdot \mathcal{R}_2 + 6 \cdot \mathcal{R}_3 + 4 \cdot \mathcal{G}$	21	-16.9168
$(592, \frac{1}{8})$	[213,70]	$1 \cdot \mathcal{H}_1 + 17 \cdot \mathcal{R}_2 + 5 \cdot \mathcal{R}_3 + 4 \cdot \mathcal{G}$	27	-20.3284
	[213,72]	$5 \cdot \mathcal{R}_3 + 7 \cdot \mathcal{R}_4 + 5 \cdot \mathcal{G}$	17	-19.8500
	[214,71]	$11 \cdot \mathcal{R}_4 + 5 \cdot \mathcal{G}$	16	-20.2712

hybrid reduction mode of LF(4) + LF2, $\varepsilon_{cp} = \varepsilon^{8^t 2^{t'}}$ is for LF(8) + LF2, and $\varepsilon_{cp} = \varepsilon^{16^t 2^{t'}}$ is for LF(16) + LF2. Note that ε_{cc} is the bias introduced in the covering code procedure, which is optimally derived according to the integer linear programming proposed in [34] with the explicit code construction of the optimal cascaded perfect codes. Given $\eta = \frac{1}{8}$, we could get the optimal cascaded perfect codes with the fixed parameters k_2 and l by Magma. In Table 5, the optimal cascaded perfect codes are presented with the parameters chosen for solving the $(512, \frac{1}{8})$, $(532, \frac{1}{8})$, $(592, \frac{1}{8})$-LPN instances. Here we use the notation $\mathcal{H}_r$ for the $[2^r - 1, 2^r - 1 - r]1$ Hamming code, $r = 1, 2, ...$, and $\mathcal{R}_e$ for the $[2e + 1, 1]e$ Repetition code, $e = 2, 3, ...$, and $\mathcal{G}$ for the $[23, 12]3$ Golay code. The notation $\mathcal{C}_1 + \mathcal{C}_2$ represents the direct sum of linear codes $\mathcal{C}_1$ and $\mathcal{C}_2$. It is shown in [34] that it needs $m = 8l \ln 2/\varepsilon_f^2$ queries to distinguish the correct guess from the others in the final solving phase. For adopting LF(k) + LF2, the number of queries is computed iteratively as $n[i-1] = (2!)^{1/2} n[i]^{1/2} 2^{b_i/2}$ for $i = t+t', ..., t+1$. And for $i = t, t-1, ..., 1$, we have $n[i-1] = 2^{3/4} n[i]^{1/4} 2^{b_i/3}$ for adopting LF(4) + LF2, and $n[i-1] = 2^{5/6} n[i]^{1/6} 2^{b_i/4}$ for adopting LF(8) + LF2, and $n[i-1] = 2^{7/8} n[i]^{1/8} 2^{3b_i/16}$ for adopting LF(16) + LF2, where $n[t+t'] = m$ and $n = k + n[0]$. According, the initial number of queries is finally derived as $N = n2^c$. Besides, the online time complexity of the LPN solving algorithm is

$$C = C_0 + \frac{PC_{11} + C_1 + C_{21} + C_{22} + C_3 + C_4 + C_5}{\Pr(w_1, k_1)},$$

with a pre-computation of $Pre = PC_{12} + PC_{21} + PC_{22} + PC_4$, and a memory complexity of about $M = nk + M_{11} + M_{12} + M_{21} + M_{22} + M_4$. We list in Table 6 the corresponding complexities when the hybrid BKW reduction modes are

adopted.

Remark 1. In the Gaussian elimination step, we suppose a to be the power of some integer q and write it as $a = q^w$. Consequently, the time complexity at Step 2 is $C_1 = (n-k)\left(a + \frac{q^w-1}{q-1}d\right)$, which is quite different from that in [34].

Remark 2. For the merging procedure of LF2, we correct the result in [34] by adding a term to represent the cost of the sorting process.

Table 6: The complexities of the 5 main steps when adopting LF(4)+LF(2), LF(8)+LF(2) and LF(16)+LF(2), respectively.

Step	Complexities
0	$C_0 = N$
1	$C_1 = (n-k)\left(a + \frac{q^w-1}{q-1}d\right)$ $M_{11} = 2^s ka, PC_{11} = (2^s - s - 1)ka$ $M_{12} = u2^{qu}, PC_{12} = \sum_{i=2}^{q} u2^{u-1}\binom{2^u-2}{i-1}$
2	LF(**4**) + LF**2** : $C_{21} = \sum_{i=1}^{t}\left(\left\lceil\frac{k+1-B_i}{e}\right\rceil n[i] + (2^{3/4}n[i]^{1/4} + 2^{1/2}n[i]^{1/2})2^{b_i/3}\right)$, $M_{21} = e2^{4e} + (2^{3/4}n[i]^{1/4} + 2^{1/2}n[i]^{1/2})2^{b_i/3}, PC_{21} = \sum_{i=2}^{4} e2^{e-1}\binom{2^e-2}{i-1}$
	LF(**8**) + LF**2** : $C_{21} = \sum_{i=1}^{t}\left(\left\lceil\frac{k+1-B_i}{e}\right\rceil n[i] + (2^{5/6}n[i]^{1/6} + 2\cdot 2^{2/3}n[i]^{1/3})2^{b_i/4}\right)$, $M_{21} = e2^{8e} + (2^{5/6}n[i]^{1/6} + 2^{2/3}n[i]^{1/3})2^{b_i/4}, PC_{21} = \sum_{i=2}^{8} e2^{e-1}\binom{2^e-2}{i-1}$
	LF(**16**) + LF**2** : $C_{21} = \sum_{i=1}^{t}\left(\left\lceil\frac{k+1-B_i}{e}\right\rceil n[i] + (2^{7/8}n[i]^{1/8}2^{3b_i/16} + 3\cdot 2^{3/4}n[i]^{1/4}2^{5b_i/24})\right)$ $M_{21} = e2^{16e} + (2^{7/8}n[i]^{1/8}2^{3b_i/16} + 2^{3/4}n[i]^{1/4}2^{5b_i/24}), PC_{21} = \sum_{i=2}^{8} e2^{e-1}\binom{2^e-2}{i-1}$
	Beside, for LF(**4**) + LF**2**, LF(**8**) + LF**2** and LF(**16**) + LF**2**, $C_{22} = \sum_{i=1}^{t'}\left(\left\lceil\frac{k+1-B_{t+i}}{f}\right\rceil n[t+i] + n[t+i-1]\right)$ $PC_{22} = f2^{f-1}(2^f - 2), M_{22} = f2^{2f}$
3	$C_3 = \sum_{i=1}^{w_1}\binom{k_1}{i}$
4	$C_4 = mh$, $M_4 = 12\cdot 2^{23}$, $PC_4 = 11\cdot 23\cdot 2^{23}$
5	$C_5 = l2^l\sum_{i=0}^{w_1}\binom{k_1}{i}$

We have found in the analysis that by adopting the above hybrid BKW reduction modes, we can get more tradeoff choices, and more significantly, we have an opportunity to reduce the memory and data complexity simultaneously for some specific LPN instances, while keeping the time complexity at a low level. We will illustrate it by solving the three core LPN instances, i.e., $(512, \frac{1}{8})$, $(532, \frac{1}{8})$, $(592, \frac{1}{8})$, in the following Section.

8 Applications and Simulations

8.1 Applications and Attacks on Schemes

Note that for all of the three LPN instances, we use a little trick when adopting the hybrid reduction mode LF(4)/LF(8)/LF(16) + LF2, i.e., we set a to be the power of 2, thus in the Gaussian elimination step, we will need to construct a table storing all the possible additions of two u-bit vectors, which will cost $PC_{12} = u2^{u-1}(2^u - 2)$ in time and $M_{12} = u2^{2u}$ in memory. Then in the collision procedure, another table is constructed storing all the possible additions of two f-bit vectors, which will cost $PC_{22} = f2^{f-1}(2^f - 2)$ in time and $M_{22} = f2^{2f}$ in memory. Let $f = u$, then we can reuse the table constructed in the Gaussian elimination procedure, resulting in $PC_{22} = 0$, $M_{22} = 0$ and meanwhile no apparent negative effect on the other complexities. For the three core LPN instances, we list in Tables 7, 8 and 9 the complexity issues when adopting the hybrid reduction modes, together with the details of the parameters chosen and the number of queries via each BKW step. We take $t = 1$ and $t' = 3$ in the collision procedure for LF(4) + LF2, $t = 1$ and $t' = 2$ for LF(8) + LF2 and $t = 1$ and $t' = 1$ for LF(16) + LF2. For LF(4)/LF(8)/LF(16), the number of bits eliminated is denoted by b, and for LF2, we take the same value for this number which is denoted by b'. Besides, each $[k_2, l]$-covering code adopted is explicitly constructed with the method proposed in [34] and thus the corresponding bias ε_{cc} introduced in this procedure is an accurate value.

Table 7: The complexity for solving the $(512, \frac{1}{8})$-LPN instance using different BKW reduction modes

Type	Parameters									Selected data $\log_2 n$
	c	$a = q^w$	u	t, b, e	t', b', f	k_1	w_1	k_2	l	
LF(4) + LF2	14	$32 = 2^5$	30	1, 112, 16	3, 66, 30	11	1	177	62	54.833
LF(8) + LF2	8	$16 = 2^4$	28	1, 187, 8	2, 65, 28	15	1	172	61	58.572
	4	$16 = 2^4$	28	1, 188, 8	2, 65, 28	18	1	172	61	58.822
LF(16) + LF2	12	$16 = 2^4$	27	1, 253, 4	1, 65, 27	8	1	174	62	56.559

Type	Data via each BKW step				
	$\log_2 n[1]$	$\log_2 n[2]$	$\log_2 n[3]$	$\log_2 n[4]$	$\log_2 n[5]$
LF(4) + LF2	66.9997	66.9995	66.9989	66.9978	-
LF(8) + LF2	65.9311	65.8623	65.7245	-	-
	65.9311	65.8623	65.7245	-	-
LF(16) + LF2	65.9681	65.9362	-	-	-

Type	Complexities ($\log_2$)			
	Time $\log_2 C$	Initial data $\log_2 N$	Memory $\log_2 M$	Pre-computation $\log_2 Pre$
LF(4) + LF2	74.751	68.833	71.492	65.183
LF(8) + LF2	74.304	66.572	69.955	59.826
	74.831	62.821	70.175	59.826
LF(16) + LF2	73.919	68.559	70.104	57.755

Table 8: The complexity for solving the $(532, \frac{1}{8})$-LPN instance using different BKW reduction modes

Type	Parameters									Selected data $\log_2 n$
	c	$a = q^w$	u	$t,\ b,\ e$	$t',\ b',\ f$	k_1	w_1	k_2	l	
LF(4) + LF2	17	$16 = 2^4$	30	$1, 115, 16$	$3, 68, 30$	11	1	185	64	56.329
LF(8) + LF2	9	$16 = 2^4$	30	$1, 193, 8$	$2, 67, 30$	15	1	181	63	60.412
	5	$16 = 2^4$	30	$1, 194, 8$	$2, 67, 30$	18	1	181	63	60.662
LF(16) + LF2	14	$16 = 2^4$	30	$1, 262, 4$	$1, 67, 30$	7	1	182	64	58.488

Type	Data via each BKW step				
	$\log_2 n[1]$	$\log_2 n[2]$	$\log_2 n[3]$	$\log_2 n[4]$	$\log_2 n[5]$
LF(4) + LF2	68.983	68.967	68.934	68.867	-
LF(8) + LF2	67.971	67.943	67.886	-	-
	67.971	67.943	67.886	-	-
LF(16) + LF2	67.903	67.805	-	-	-

Type	Complexities ($\log_2$)			
	Time	Initial data	Memory	Pre-computation
LF(4) + LF2	76.861	73.329	73.370	65.183
LF(8) + LF2	76.424	69.412	71.934	63.908
	76.935	65.662	72.175	63.908
LF(16) + LF2	75.992	72.488	72.386	63.907

Table 9: The complexity for solving the $(592, \frac{1}{8})$-LPN instance using different BKW reduction modes

Type	Parameters									Selected data $\log_2 n$
	c	$a = q^w$	u	$t,\ b,\ e$	$t',\ b',\ f$	k_1	w_1	k_2	l	
LF(4) + LF2	15	$32 = 2^5$	34	$1, 129, 16$	$3, 75, 34$	10	1	213	70	62.744
LF(8) + LF2	5	$16 = 2^4$	34	$1, 213, 8$	$2, 75, 34$	10	1	214	71	66.739
LF(16) + LF2	12	$16 = 2^4$	30	$1, 290, 4$	$1, 74, 30$	3	1	213	72	64.619

Type	Data via each BKW step				
	$\log_2 n[1]$	$\log_2 n[2]$	$\log_2 n[3]$	$\log_2 n[4]$	$\log_2 n[5]$
LF(4) + LF2	75.978	75.955	75.910	75.820	-
LF(8) + LF2	75.931	75.863	75.726	-	-
LF(16) + LF2	74.952	74.904	-	-	-

Type	Complexities ($\log_2$)			
	Time	Initial data	Memory	Pre-computation
LF(4) + LF2	83.792	77.744	81.531	73.357
LF(8) + LF2	83.599	71.738	79.387	72.088
LF(16) + LF2	82.889	76.619	79.926	63.907

The comparison of our LPN solving algorithms with the previous ones in [18, 10] are presented in Table 10 on the $(512, \frac{1}{8})$, $(532, \frac{1}{8})$ and $(592, \frac{1}{8})$-LPN

instances. Note that we have derived more tradeoff choices by adopting the newly proposed hybrid reduction modes, i.e., LF(4)/LF(8)/LF(16) + LF2, on the basis of the single-list k-sum problem. With the carefully chosen attack parameters, our algorithm can have clear advantages over the previous best ones. First, on one hand, our algorithm of solving the $(512, \frac{1}{8})$-LPN instance can get at least 2^7 times lower than that in [18] with less data and memory. On the other hand, our algorithm is about 2^5 times faster than that in [10], while the data and memory complexity are kept almost at the same level. Second, our results for solving $(532, \frac{1}{8})$-LPN instance are better than that in [10] by a factor of about 2^5, and better than that in [18] by a factor of 2^{14}, depending on the parameters chosen. Finally, for solving the $(592, \frac{1}{8})$-LPN, we have derived some tradeoff choices with the time complexity 2^5 times lower than that in [10] and 2^{14} times lower than that in [18].

Table 10: Comparison of different LPN solving algorithms with the three core LPN instances, i.e., $(512, \frac{1}{8})$,$(532, \frac{1}{8})$,$(592, \frac{1}{8})$-LPN instances

(k, η)-LPN	Complexities ($\log_2$)	Algorithm					
		[18]	[10]	**ours**	**ours**	**ours**	**ours**
$(512, \frac{1}{8})$	Time	82.27	79.37	74.75	74.30	74.83	73.92
	Data	63.60	62.20	68.83	66.57	62.82	68.56
	Memory	72.60	71.20	71.49	69.96	70.18	70.10
$(532, \frac{1}{8})$	Time	90.43	81.64	76.86	76.42	76.94	75.99
	Data	-	-	73.33	69.41	65.66	72.49
	Memory	-	-	73.37	71.93	72.18	72.39
$(592, \frac{1}{8})$	Time	97.87	88.25	83.79	83.60	82.89	-
	Data	-	-	77.74	71.74	76.62	-
	Memory	-	-	81.53	79.39	79.93	-

Note that the $(512, \frac{1}{8})$-LPN instance is widely adopted in various LPN-based cryptosystems, e.g., HB$^+$ [23], HB$^\#$ [14] and LPN-C [15]. With the same arguments as those in Section 6.3 of [34], we could successfully launch concrete attacks on these protocols. The $(532, \frac{1}{8})$-LPN instance is adopted in Lapin [20] for 80-bit security; unfortunately, our analysis has shown that this is unreachable. We have to change the parameter configuration in the circumstances where such instances are intended to be used. At last, our attacks have indicated as well that the $(592, \frac{1}{8})$-LPN instance has a very thin security margin for 80-bit security.

8.2 Experimental Results

To verify the validity of the new algorithms, we have conducted extensive simulations on the reduced $(100, \frac{1}{8})$, $(128, \frac{1}{8})$-LPN instances, respectively. The experiments have been fully implemented in C language on one core of a single PC, running with Windows 7, Intel Core i3-2120 CPU @ 3.30 GHz and 4.00GB RAM. In general, the experimental results match the theoretical analysis quite

well.

$(100, \frac{1}{8})$-LPN instance. For this instance, we have the following set of parameters and the corresponding complexities results. The number of initial queries is chosen to be $N = 2^{22.49}$.

- In Step 0, set $c = 6$; accordingly, the number of queries remained is $n = 2^{16.49}$, and the effective size of the secret is changed to $k = 100 - c = 94$.
- In Step 1, set $a = 2^3$ and $u = 8$. After this step, the number of queries available is $n[0] = n - k \approx 2^{16.49}$.
- In Step 2, adopt the hybrid BKW reduction mode LF(4)+LF2. Set $(t, b, e) = (1, 33, 5)$ for LF(4), then the number of queries remained is $n[1] = 2^{18.95}$ and the effective size of the secret is reduced to $k = 61$ after the LF(4) reduction. Further set $(t', b', f) = (1, 18, 8)$ for LF2, then the number of queries remained is $n[2] = 2^{18.90}$ and the effective size of the secret is reduced to $k = 43$ after the LF2 reduction. After this step, the bias of the noise is reduced to $\varepsilon_{cp} = (\frac{3}{4})^8$.
- In Step 3, set $k_1 = 5$ and $w_1 = 1$. Thus the number of queries remains unchanged, and the effective size of the secret is reduced to $k_2 = k - k_1 = 38$.
- In Step 4, set $k_2 = 38$, $l = 16$, and construct explicitly the $[38, 16]$-covering code by concatenating one $\mathcal{G} = [23, 12]$ Golay code, one repetition code $\mathcal{R}_2 = [5, 1]$, three Hamming code $\mathcal{H}_2 = [3, 1]$ and one Hamming code $\mathcal{H}_1 = [1, 0]$, i.e., $1 \cdot \mathcal{G} + 1 \cdot \mathcal{R}_2 + 3 \cdot \mathcal{H}_2 + 1 \cdot \mathcal{H}_1$. The bias introduced in this procedure is explicitly computed as $\varepsilon_{cc} = 2^{-2.8902}$. After this step, the number of queries remains the same and the effective size of the secret is reduced to $l = 16$. Besides, the final bias of the noise is $\varepsilon_f = \varepsilon_{cp}\varepsilon_{cc} = 2^{-6.2105}$.
- In Step 5, the number of queries remained is $m = 2^{18.90}$. Note that we have chosen $m > 8l \ln 2/\varepsilon_f^2$ $(= 2^{18.8922})$, thus the success probability for the recovery is quite close to 1, as illustrated in [34] and better than that in [10].

For the above set of parameters, the overall complexity is $C = 2^{25.32}$ in time, $N = 2^{22.49}$ for initial data, $M = 2^{26.85}$ in memory. Currently, our non-optimized implementation runs for tens of minutes to recover the targeted secret bits.

$(128, \frac{1}{8})$-LPN instance. For this instance, we list a set of parameters and the corresponding complexities results as follows. The number of initial queries is chosen to be $N = 2^{25.61}$.

- In Step 0, set $c = 5$. Thus, the number of queries remained is $n = 2^{20.61}$, and the effective size of the secret is changed to $k = 128 - c = 123$.
- In Step 1, set $a = 2^3$ and $u = 9$. After this step, the number of queries available is $n[0] = n - k \approx 2^{20.61}$.
- In Step 2, adopt the hybrid BKW reduction mode LF(8) + LF2, i.e., set $(t, b, e) = (1, 64, 3)$ for LF(8), then the number of queries remained is $n[1] = 2^{22.67}$ and the effective size of the secret is reduced to $k = 59$ after the LF(8) reduction. Further set $(t', b', f) = (1, 22, 9)$ for LF2, then the number of queries remained is $n[2] = 2^{22.35}$ and the effective size of the secret is reduced to $k = 37$ after the LF2 reduction. After this step, the bias of the noise is reduced to $\varepsilon_{cp} = (\frac{3}{4})^{16}$.

– In Step 3, set $k_1 = 7$ and $w_1 = 1$. Thus the number of queries remains unchanged, and the effective size of the secret is reduced to $k_2 = k - k_1 = 30$.
– In Step 4, set $k_2 = 30$, $l = 19$, and construct explicitly the $[30, 19]$-covering code by concatenating one $\mathcal{G} = [23, 12]$ Golay code and one trivial code $\mathcal{Z}_2^7 = [7, 7]$. The bias introduced in this procedure is explicitly computed as $\varepsilon_{cc} = 2^{-1.1739}$. After this step, the number of queries remains the same and the effective size of the secret is reduced to $l = 19$. Besides, the final bias of the noise is $\varepsilon_f = \varepsilon_{cp}\varepsilon_{cc} = 2^{-7.8145}$.
– In Step 5, the number of queries remained is $m = 2^{22.35}$. Note that we have set $m > 8l \ln 2/\varepsilon_f^2$ $(= 2^{22.3485})$, thus the success probability is quite close to 1.

For the above set of parameters, the overall complexity is $C = 2^{29.50}$ in time, $N = 2^{25.61}$ for the initial data, $M = 2^{28.35}$ in memory. So far, it takes a few hours for our non-optimized implementation to restore the targeted bits in the secret vector.

9 Conclusions

In this paper, we have presented new algorithmic improvements to the LPN solving problem. The first one, efficient algorithms for the single list k-sum problem over a binary field and an arbitrary finite field, are the first answers to the corresponding open problem of Minder and Sinclair and would probably have other applications in some other areas. Based on this algorithm, the second technique, the hybrid mode of BKW reduction could make good use of different kinds of reductions without increasing the overall complexity and achieve better tradeoffs which are impossible previously. Third, we have improved the dominate matrix multiplication in the Gaussian elimination step which is the bottleneck of the previous LPN solving algorithms. Finally, we integrated the above three new techniques into the refined framework for solving LPN, which is further applied to the core LPN instances studied in the open literature. The new algorithms yielded remarkable complexity reductions compared to all the previously known methods and the experimental results clearly show the gain on efficiency that the new techniques bring to the LPN solving algorithms. It is safe to say now that the 80-bit security bound of the instances suggested in cryptographic schemes like HB^+, $\text{HB}^\#$, LPN-C and Lapin cannot be reached following the current parameters, it is advised to update the parameter configuration in these schemes to get the expected security level.

References

1. Albrecht, M. R., Bard, G. V., Hart, W., Algorithm 898: efficient multiplication of dense matrices over GF(2). *ACM Trans. Math. Softw.*, 37(1), <http://dx.doi.org/10.1145/1644001.1644010>

2. Applebaum, B., Cash., D., Peikert, C. and Sahai, A., Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Halevi. S., editor, *Advances in Cryptology–CRYPTO 2009*, LNCS vol. 5677, pages 595–618. Springer Berlin Heidelberg, 2009.
3. Becker, A., Joux, A., May, A., and Meurer A. Decoding random binary linear codes in $2^{n/20}$: how $1 + 1 = 0$ improves information set decoding. In Pointcheval. D. and Johansson. T., eds, *Advances in Cryptology–EUROCRYPT 2012*, LNCS vol. 7237, pages 520–536. Springer Berlin Heidelberg, 2012.
4. Bernstein, D.J., Lange, T., and Peters C. Smaller decoding exponents: ball-collision decoding. In Rogaway, P. editor, *Advances in Cryptology–CRYPTO 2011*, LNCS vol. 6841, pages 743–760. Springer Berlin Heidelberg, 2011.
5. Bernstein, D.J., Optimizing linear maps modulo 2, available at `http://binary.cr.yp.to/linearmod2-20090830.pdf`.
6. Bernstein, D.J., Lange, T. Never trust a bunny. In Hoepman, J.-H., Verbauwhede, I. (eds.) *Radio Frequency Identification, Security and Privacy–RFIDSec 2012*. LNCS vol. 7739, pages 137–148. Springer, Heidelberg, 2013.
7. Blum, A., Kalai, A., Wasserman, H. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM* 50(4), pages 506–519, 2003.
8. Blum, A. Frust, M.L., Kearns, M.J., and Lipton. R.J. Cryptographic primitives based on hard learning problems. In Stinson, D.R., editor, *Advances in Cryptology–CRYPTO 1993*, LNCS vol. 773, pages 278–291. Springer, Heidelberg, 1994.
9. Bogos, S., Vaudenay, S. Observations on the LPN solving algorithm from Eurocrypt'16. available at `https://eprint.iacr.org/2016/437.pdf`
10. Bogos, S., Vaudenay, S. Optimization of LPN solving algorithms. In Cheon J. H., Takagi. T, (eds.) *Advances in Cryptology–ASIACRYPT 2016*, Part 1, LNCS vol. 10031, pages 703–728. Springer, Berlin Heidelberg, 2016.
11. Chan, T. M., Speeding up the four russians algorithm by about one more logarithmic factor. *ACM-SIAM Symposium on Discrete Algorithms–SODA 2015*, pages. 212–217.
12. `https://en.wikipedia.org/wiki/Model_of_computation`
13. Feldman, V., Gopalan, P., Khot S. and Ponnuswami A. K., New results for learning noisy parities and halfspaces. *Proceeding of the 47th Annual IEEE Symposium on Foundations of Computer Scicence–FOCS 2006*, pages 563–574. IEEE Computer Society, Washington, DC, USA, 2006.
14. Gilbert, H., Robshaw, M., and Seurin. Y. $\text{HB}^{\#}$: Increasing the security and efficiency of HB^{+}. In N. Smart, editor, *Advances in Cryptology–EUROCRYPT 2008*, LNCS vol. 4965, pages 361–378. Springer Berlin Heidelberg, 2008.
15. Gilbert, H., Robshaw, M., and Seurin. Y. How to encrypt with the LPN problem. In L. Aceto, I. Damgård, L. Goldberg, M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *Automata, Languages and Programming–ICALP' 2008*, LNCS vol. 5126, pages 679–690. Springer Berlin Heidelberg, 2008.
16. Gilbert, H., Robshaw, M., and Sibert. H. Active attack against HB^{+}: a provably secure lightweight authentication protocol. *Electronics Letters*, 41(21): pages 1169–1170, 2005.
17. Goldreich, O., Goldwasser, S., Micali, S. How to construct random functions. *Journal of the ACM*, 33: pages 792–807, 1986.
18. Guo, Q., Johansson, T., Löndahl, C. Solving LPN using covering codes. In: Sarkar, P., Iwata, T. (eds.) *Advances in Cryptology–ASIACRYPT 2014*. LNCS vol. 8873, pages. 1–20. Springer, Heidelberg 2014.

19. Hastad, J., Impagliazzo, R., Levin, L.A., and Luby, M. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4): pages 1364–1396, 1999.
20. Heyse, S., Kiltz, E., Lyubashevsky. V., Paar, C. and Pietrzak. K. Lapin: An efficient authentication protocol based on ring-LPN. In Canteaut, A., editor, *Fast Software Encryption–FSE'2012*, LNCS vol. 7549, pages 346–365. Springer Berlin Heidelberg, 2012.
21. Hopper N. and Blum. M., Secure human identification protocols. In Boyd, C., editor, *Advances in Cryptology–ASIACRYPT 2001*, LNCS vol. 2248, pages 52–66. Springer Berlin Heidelberg, 2001.
22. Jain., A., Pietrzak, K. and Tentes, A. Commitments and efficient zero-knowledge from hard learning problems. In Wang. X and Sako. K, eds, *Advances in Cryptology–ASIACRYPT 2012*, LNCS vol. 7658, pages 663–680. Springer Berlin Heidelberg, 2012.
23. Juels, A. and Weis., S. Authenticating pervasive devices with human protocols. In Shoup. V., editor, *Advances in Cryptology–CRYPTO 2005*, LNCS vol. 3621, pages 293–308. Springer Berlin Heidelberg, 2005.
24. Katz, J., Shin, J.S. and Smith, A.,. Parallel and concurrent security of the HB and HB^+ protocols. *Journal of Cryptology*, volume 23(3), pages 402–421, 2010.
25. Kiltz, E., Pietrzak, K., Cash, D., Jain, A. and Venturi., D. Efficient authentication from hard learning problems. In Paterson K., editor, *Advances in Cryptology–EUROCRYPT 2011*, LNCS vol. 6632, pages 7–26. Springer Berlin Heidelberg, 2011.
26. Kirchner, P.: Improved generalized birthday attack. available at `http://eprint.iacr.org/2011/377.pdf`
27. Levieil, É. and Fouque, P.-A. An improved LPN algorithm. In: De Prisco, R., Yung, M. (eds.) Security and Cryptography for Networks–SCN 2006. LNCS vol. 4116, pages 348–359. Springer, Heidelberg 2006.
28. Luby, M. and Rackoff, C. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2): pages 373–386, 1988.
29. May. A., Meurer, A., Thomae E. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In Lee. D.H. and Wang X., eds, *Advances in Cryptology–ASIACRYPT 2011*, LNCS vol. 7073, pages 107–124. Springer Berlin Heidelberg, 2011.
30. May., A. and Ozerov, I. On computing nearest neighbors with applications to decoding of binary linear codes. In Oswald. E. and Fischlin. M, eds, *Advances in Cryptology–EUROCRYPT 2015*, LNCS vol. 9056, pages 203–228. Springer Berlin Heidelberg, 2015.
31. Minder, L., Sinclair, A. The extended k-tree algorithm. *Journal of Cryptology*, vol. 25(2), pages. 349–382 2012.
32. Stern., J. A new identication scheme based on syndrome decoding. In Stinson. D.R. editor, *Advances in Cryptology–CRYPTO 1993*, LNCS vol. 773, pages 13–21. Springer, Heidelberg, 1994.
33. Wagner., D. A generalized birthday problem. In Yung., M. editor, *Advances in Cryptology–CRYPTO 2002*, LNCS vol. 2442, pages 288–304. Springer Berlin Heidelberg, 2002.
34. Zhang, B., Jiao, L., and Wang, M.S. Faster algorithm for solving LPN. In Fischlin, M., Coron, J. (eds.) *Advances in Cryptology–EUROCRYPT 2016*, LNCS vol. 9665, pages. 168–195. Springer, Heidelberg 2016.

A Proof of Theorem 1

Proof. We first consider the linear program by relaxing the integrality constraint.

First, to show the feasibility of the linear program, we set $l_1 = ... = l_{p-1} = 0$ and $l_p = n$, then we have $a_i = 2^i a_0 - (2^i - 1)$ for $i = 1, ..., p-1$, and $a_p = 2^p a_0 - n - (2^p - 1)$. In this case, we set $u = \max\limits_{0 \le i \le p-1} a_i = 2^{p-1} a_0 - (2^{p-1} - 1)$ and thus the constraint $a_i \le u,\ i = 0, 1, ..., p-1$ is clearly satisfied. We also have $l_i \ge 0$ are integers for $i = 1, ..., p$ and $\sum_{i=1}^{p} l_i \ge n$. For the constraint condition $a_p \ge s$, it translates to $2^p a_0 \ge n + s + (2^p - 1)$ and further $a_0 \ge \frac{n+s}{2^p} + \frac{2^p - 1}{2^p}$, which is equivalent to the condition (1). Hence we know that there is indeed a solution and the linear program is feasible.

Next, let us show that any solution not of the given form can be strictly improved. Consider first a feasible solution $\boldsymbol{l} = (l_1, ..., l_p)$ in which there exists some index $i \in \{1, ..., p-1\}$ such that $l_i > 0$ and $a_i < u$. Let $\varepsilon > 0$ be a suitably small integer, and define $\boldsymbol{l'} = (l'_1, ..., l'_{i-1}, l'_i, l'_{i+1}, ..., l'_p) = (l_1, ..., l_{i-1}, l_i - \varepsilon, l_{i+1} + 2\varepsilon, ..., l_p)$. In this case, a_i is changed to $a'_i = a_i + \varepsilon$. It's easy to see that $\boldsymbol{l'}$ is another feasible solution with the same value of u, and we have $\sum_{i=1}^{p} l'_i = \sum_{i=1}^{p} l_i - \varepsilon + 2\varepsilon \ge n + \varepsilon > n$. Similarly, consider another feasible solution $\boldsymbol{l}$ which has $a_p > s$ and let us define $\boldsymbol{l'} = (l_1, ..., l_{p-1}, l_p + \varepsilon)$. In this case, a_p is changed to $a_p - \varepsilon$. Note that $\boldsymbol{l'}$ is another feasible solution with the same value of u and also $\sum_{i=1}^{p} l'_i > n$. From the above analysis, we know that any solution $\boldsymbol{l}$ that does not satisfy the conditions $l_i = 0$ or $a_i = u$ for all $i = 1, ..., p-1$ and $a_p = s$, can be transformed into another solution $\boldsymbol{l'}$ with the same value of the objective function u which does satisfy these conditions, i.e., $l'_i = 0$ or $a'_i = u$ for all $i = 1, ..., p-1$ and $a'_p = s$, and also satisfy $\sum_{i=1}^{p} l'_i > n$. In the following, we will show that such a solution $\boldsymbol{l'}$ can be further transformed into another solution $\boldsymbol{l''}$ with a smaller value of u, i.e., the maximal a'_i for the solution $\boldsymbol{l'}$ can be reduced. We first emphasize that $a'_0 \ne u$. Suppose $a'_0 = u$, then $a'_i \le a'_0$ for all $i = 1, ..., p-1$. From Eq.(2) we have $l'_i = 2a'_{i-1} - a'_i - 1$, then $\sum_{i=1}^{p} l'_i = \sum_{i=1}^{p} (2a'_{i-1} - a'_i - 1) > n$, or equivalently, $2a'_0 + \sum_{i=1}^{p-1} a'_i - a'_p - p > n$, and hence $(1+p)a'_0 - s > n + p$, which contradicts with condition (3), so we have $a'_0 \ne u$. Now let j be an index such that $a'_j = u$, define $\boldsymbol{l''} = (l'_1, ..., l'_{j-1}, l'_j + \varepsilon, l'_{j+1} - 2\varepsilon, ..., l'_p)$, then $a''_j = a'_j - \varepsilon$ and $\sum_{i=1}^{p} l''_i = \sum_{i=1}^{p} l'_i - \varepsilon$. Now we will show that $\boldsymbol{l''}$ is a feasible solution. To prove this, we only need to check that $l'_{j+1} > 0$. Let us suppose $l'_{j+1} = 0$, then $a'_{j+1} = 2a'_j - 1 = 2u - 1$. If $j + 1 < p$, we have $a'_{j+1} = 2u - 1 > u$, which conflicts with the condition $a'_{j+1} \le u$; If $j + 1 = p$, we have $s = a'_p = a'_{j+1} = 2u - 1 > 2a'_0 - 1$, which violates our assumption that $s \le 2 \log m - 1$.

All the above shows that any optimal solution must satisfy $l_i = 0$ or $a_i = u$ for all $i = 1, ..., p-1$ and $a_p = s$. Besides, we emphasize that the indices j for which $l_j = 0$ must form an initial segment. (For if not, suppose $a_j = u$ and $l_{j+1} = 0$, then $a_{j+1} = 2u - 1 > u$ which conflicts with the condition $a_{j+1} \le u$.) We denote r to be the intermediate index, i.e., we have $l_i = 0$ for $1 \le i < r$ and $a_i = u$ for $r \le i \le p-1$. Up to now, we can compute the following using Equation (2) as follows: Since $l_i = 0$ for $1 \le i < r$, we have $a_i = 2^i a_0 - (2^i - 1)$; Since $a_r = u$, we have $l_r = 2a_{r-1} - a_r - 1 = 2^r a_0 - u - (2^r - 1)$; Since $a_i = u$ for $r < i \le p-1$, we have $l_i = u - 1$; Since $a_p = s$, we have $l_p = 2a_{p-1} - a_p - 1 = 2u - s - 1$.

Now it remains to determine the value of r and u. From $a_{r-1} \leq u$ and $0 \leq l_r = 2^r a_0 - u - (2^r - 1)$ we have $2^{r-1}a_0 - (2^{r-1} - 1) \leq u \leq 2^r a_0 - (2^r - 1)$. Note that the constraint condition $\sum_{i=1}^{p} l_i \geq n$ must be tight in an optimal solution, we obtain $n = \sum_{i=1}^{p} l_i = (p-r)u + (2^r a_0 - (2^r - 1)) - (p-r) - s$. Further, we have

$$\begin{aligned} n &\leq (p-r)(2^r a_0 - (2^r - 1)) + (2^r a_0 - (2^r - 1)) - (p-r) - s \\ &= (p-r+1)2^r(a_0 - 1) - s + 1 \\ n &\geq (p-r)(2^{r-1} a_0 - (2^{r-1} - 1)) + (2^r a_0 - (2^r - 1)) - (p-r) - s \\ &= (p-(r-1)+1)2^{r-1}(a_0 - 1) - s + 1 \end{aligned}$$

The intermediate index r is defined as the least integer such that $n \leq (p-r+1)2^r(a_0-1) - s + 1$. Once knowing p, we can compute u from $n = (p-r)u + (2^r a_0 - (2^r - 1)) - (p-r) - s$ as $u = \frac{n+s+(2^r-1)-2^r a_0}{p-r} + 1$. Up to now, we have derived the optimal solution to the linear program. However, this optimal solution for l_i may not be integers. Fortunately, the optimal solution $l_1, ..., l_p$ of the integer program can be obtained by replacing u by $\lceil u \rceil$ similarly to the rounding argument in [31]. This completes the proof. □

B Schematic of addition of $a = 3^w$ u-bit vectors

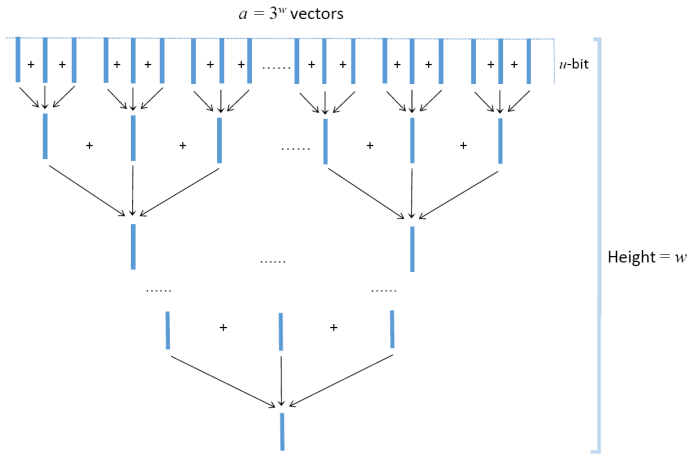

Fig. 3: Schematic of addition of $a = 3^w$ u-bit vectors