Faster Algorithms for Solving LPN

Bin Zhang[1,2], Lin Jiao[1,3], and Mingsheng Wang[4]

[1]TCA Laboratory, SKLCS, Institute of Software,
Chinese Academy of Sciences, Beijing 100190, China
[2]State Key Laboratory of Cryptology, P.O.Box 5159, Beijing 100878, China
[3]University of Chinese Academy of Sciences, Beijing 100049, China
[4]State Key Laboratory of Information Security, Institute of Information Engineering,
Chinese Academy of Sciences, Beijing 100093, China
{zhangbin,jiaolin}@tca.iscas.ac.cn

Abstract. The LPN problem, lying at the core of many cryptographic constructions for lightweight and post-quantum cryptography, receives quite a lot attention recently. The best published algorithm for solving it at Asiacrypt 2014 improved the classical BKW algorithm by using covering codes, which claimed to marginally compromise the 80-bit security of HB variants, LPN-C and Lapin. In this paper, we develop faster algorithms for solving LPN based on an optimal precise embedding of cascaded concrete perfect codes, in a similar framework but with many optimizations. Our algorithm outperforms the previous methods for the proposed parameter choices and distinctly break the 80-bit security bound of the instances suggested in cryptographic schemes like HB^+, $HB^\#$, LPN-C and Lapin.

Keywords: LPN, BKW, Perfect code, HB, Lapin

1 Introduction

The Learning Parity with Noise (LPN) problem is a fundamental problem in modern cryptography, coding theory and machine learning, whose hardness serves as the security source of many primitives in lightweight and post-quantum cryptography. It is closely related to the problem of decoding random linear codes, which is one of the most important problems in coding theory, and has been extensively studied in the last half century.

In the LPN problem, there is a secret $\mathbf{x} \in \{0,1\}^k$ and the adversary is asked to find $\mathbf{x}$ given many noisy inner products $\langle \mathbf{x}, \mathbf{g} \rangle + e$, where each $\mathbf{g} \in \{0,1\}^k$ is a random vector and the noise e is 1 with some probability η deviating from $1/2$. Thus, the problem is how to efficiently restore the secret vector given some amount of noisy queries of the inner products between itself and certain random vectors.

The cryptographic schemes based on LPN are appealing both for theoretical and practical reasons. The earliest proposal dated back to the HB, HB^+, $HB^\#$ and AUTH authentication protocols [12, 18–20]. While HB is a minimalistic protocol secure in a passive attack model, the modified scheme HB^+ with one extra

round is found to be vulnerable to active attacks, i.e., man-in-the-middle attacks [14]. HB$^{\#}$ was subsequently proposed with a more efficient key representation using a variant called Toeplitz-LPN. Besides, there is also a message encryption scheme based on LPN, i.e., the LPN-C scheme in [13] and some message authentication codes (MACs) using LPN in [10, 20], allowing for constructions of identification schemes provably secure against active attacks. Another notable scheme, Lapin, was proposed as a two-round identification protocol [16], based on the LPN variant called Ring-LPN where the samples are elements of a polynomial ring. Recently, an LPN-based encryption scheme called Helen was proposed with concrete parameters for different security levels [11].

It is of primordial importance to study the best possible algorithms that can efficiently solve the LPN problem. The seminal work of Blum et al. in [5], known as the BKW algorithm, employs an iterated collision procedure of the queries to reduce the dependency on the information bits with a folded noise level. Levieil and Fouque proposed to exploit the Fast Walsh-Hadamard (FWHT) Transform in the process of searching for the secret in [22]. They also provided different security levels achieved by different instances of LPN, which are referenced by most of the work thereafter. In [21], Kirchner suggested to transform the problem into a systematic form, where each secret bit appears as an observed symbol perturbed by noise. Then Bernstein and Lange demonstrated in [4] the utilization of the ring structure of Ring-LPN in matrix inversion to further reduce the attack complexity, which can be applied to the common LPN instances by a slight modification as well. None of the above algorithms manage to break the 80-bit security of Lapin, nor the parameters suggested in [22] as 80-bit security for LPN-C [13]. At Asiacrypt 2014, a new algorithm for solving LPN was presented in [15] by using covering codes. It was claimed that the 80-bit security bound of the common $(512, 1/8)$-LPN instance can be broken within a complexity of $2^{79.7}$, and so do the previously unbroken parameters of HB variants, Lapin and LPN-C[1]. It shared the same beginning steps of Gaussian elimination and collision procedure as that in [4], followed by the covering code technique to further reduce the dimension of the secret with an increased noise level, also it borrowed the well known Walsh Transform technique from fast correlation attacks on stream ciphers [2, 7, 23], renamed as subspace hypothesis testing.

In this paper, we propose faster algorithms for solving LPN 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. Our new technique is generic and can be applied to any given (k, η)-LPN instance, while in [15] the code construction methods for covering are missing and only several specific parameters for $(512, 1/8)$-LPN instance were given in their presentation at Asiacrypt 2014. From the explicit covering, we can derive the bias introduced by covering in our construction *accurately*, and derive the attack complexity precisely. It is shown that following some tradeoff techniques,

[1] The authors of [15] redeclared their results in their presentation at Asiacrypt 2014, for the original results are incorrect due to an insufficient number of samples used to learn an LPN secret via Walsh-Hadamard Transform.

appropriate optimization of the algorithm steps in a similar framework as that in [15] can reduce the overall complexity further. We begin with a theoretical justification of the experimental results on the current existing BKW algorithms, and then propose a general form of the BKW algorithm which exploits tuples in collision procedure with a simulation verification. We also propose a technique to overcome the data restriction efficiently based on goodness-of-fit test using χ^2-statistic both in theory and experiments. In the process, we theoretically analyze the number of queries needed for making a reliable choice for the best candidate and found that the quantity $8l\ln 2/\epsilon_f^2$ is much more appropriate when taking a high success probability into account[2], where ϵ_f is the bias of the final approximation and l is the bit length of the remaining secret information. We also provide the terminal condition of the solving algorithm, correct an error that may otherwise dominate the complexity of the algorithm in [15] and push the upper bound up further, which are omitted in [15]. We present the complexity analysis of the improved algorithm based on three BKW types respectively, and the results show that our algorithm well outperforms the previous ones. Now it is the first time to distinctly break the 80-bit security of both the $(512, 1/8)$- and $(532, 1/8)$- LPN instances, and the complexity for the $(592, 1/8)$-instance just slightly exceeds the bound. A complexity comparison of our algorithm with the previous attacks is shown in Table 1. More tradeoff choices are possible and can be found in Section 6.2.

Table 1. Comparison of different algorithms with the instance $(512, 1/8)$

Algorithm	Complexities ($\log_2$)		
	Data	Memory	Time
Levieil-Fouque [22]	75.7	84.8	87.5
Bernstein-Lange [4]	68.6	77.6	85.8
Corrected [15]	63.6	72.6	79.7[1]
This paper	63.5	68.2	72.8

[1] The number of queries we chosen is the twice as that presented for correction in the presentation at Asiacrypt 2014 to assure a success probability of almost 1. Note that if the same success probability is achieved, the complexity of the attack in [15] will exceed the 2^{80} bound.

This paper is organized as follows. We first introduce some preliminaries of the LPN problem in Section 2 with a brief review of the BKW algorithm. In Section 3, a short description of the algorithm using covering codes in [15] is presented. In Section 4, we present the main improvements and more precise data complexity analysis of the algorithm for solving LPN. Then we propose and analyze certain BKW techniques in Section 5. In Section 6, we complete the faster algorithm with more specific accelerated techniques at each step, together with the applications to the various LPN-based cryptosystems. Finally, some conclusions are provided in Section 7.

[2] The authors of [15] have chosen $4l\ln 2/\epsilon_f^2$ to correct the original estimate of $1/\epsilon_f^2$ as the number of queries in their presentation at Asiacrypt 2014.

2 Preliminaries

In this Section, some basic notations of the LPN problem are introduced with a review of the BKW algorithm that is relevant to our analysis later.

2.1 The LPN Problem

Definition 1 (LPN problem). *Let Ber_η be the Bernoulli distribution, i.e., if $e \leftarrow Ber_\eta$ then $Pr[e=1]=\eta$ and $Pr[e=0]=1-\eta$. Let $\langle \mathbf{x}, \mathbf{g} \rangle$ denote the scalar product of the vectors $\mathbf{x}$ and $\mathbf{g}$, i.e., $\mathbf{x} \cdot \mathbf{g}^T$, where $\mathbf{g}^T$ denotes the transpose of $\mathbf{g}$. Then an LPN oracle $\Pi_{LPN}(k,\eta)$ for an unknown random vector $\mathbf{x} \in \{0,1\}^k$ with a noise parameter $\eta \in (0, \frac{1}{2})$ returns independent samples of*

$$(\mathbf{g} \xleftarrow{\$} \{0,1\}^k, e \leftarrow Ber_\eta : \langle \mathbf{x}, \mathbf{g} \rangle + e).$$

The (k,η)-LPN problem consists of recovering the vector $\mathbf{x}$ according to the samples output by the oracle $\Pi_{LPN}(k,\eta)$. An algorithm $\mathcal{S}$ is called (n,t,m,δ)-solver if $Pr[\mathcal{S} = \mathbf{x} : \mathbf{x} \xleftarrow{\$} \{0,1\}^k] \geq \delta$, and runs in time at most t and memory at most m with at most n oracle queries.

This problem can be rewritten in a matrix form as $\mathbf{z} = \mathbf{x}\mathbf{G} + \mathbf{e}$, where $\mathbf{e} = [e_1\ e_2\ \cdots\ e_n]$ and $\mathbf{z} = [z_1\ z_2\ \cdots\ z_n]$, each $z_i = \langle \mathbf{x}, \mathbf{g}_i \rangle + e_i, i = 1, 2, \ldots, n$. The $k \times n$ matrix $\mathbf{G}$ is formed as $\mathbf{G} = [\mathbf{g}_1^T\ \mathbf{g}_2^T\ \cdots\ \mathbf{g}_n^T]$. Note that the cost of solving the first block of the secret vector $\mathbf{x}$ dominates the total cost of recovering $\mathbf{x}$ according to the strategy applied in [1].

Lemma 1 (Piling-up Lemma). *Let $X_1, X_2, \ldots, X_n$ be independent binary random variables where each $Pr[X_i = 0] = \frac{1}{2}(1+\epsilon_i)$, for $1 \leq i \leq n$. Then,*

$$Pr[X_1 + X_2 + \cdots + X_n = 0] = \frac{1}{2}(1 + \prod_{i=1}^{n} \epsilon_i).$$

2.2 The BKW Algorithm

The BKW algorithm is proposed in the spirit of the generalized birthday algorithm [25], working on the columns of $\mathbf{G}$ as

$$\mathbf{g}_i + \mathbf{g}_j = [*\ *\ \cdots\ *\ \underbrace{0\ 0\ \cdots\ 0}_{b}], \text{ and } (z_i + z_j) = \mathbf{x}(\mathbf{g}_i^T + \mathbf{g}_j^T) + (e_i + e_j),$$

which iteratively reduces the effective dimension of the secret vector. Let the bias ϵ be defined by $\Pr[e=0] = \frac{1}{2}(1+\epsilon)$, then $\Pr[e_i + e_j = 0] = \frac{1}{2}(1+\epsilon^2)$ according to the pilling-up lemma. Formally, the BKW algorithm works in two phases: reduction and solving. It applies an iterative sort-and-merge procedure to the queries and produces new entries with the decreasing dimension and increasing noise level; finally it solves the secret by exhausting the remaining and test the presence of the expected bias. The framework is as follows.

Algorithm 1 Framework of the BKW Algorithm

Input: The $k \times n$ matrix $\mathbf{G}$ and received $\mathbf{z}$, the parameters b, t.

1: Put the received vector as a first row in the matrix, $\mathbf{G}_0 \leftarrow \begin{bmatrix} \mathbf{z} \\ \mathbf{G} \end{bmatrix}$.

Reduction phase:

2: **for** $i = 1$ to t **do**
3: **Sorting**: Partition the columns of $\mathbf{G}_{i-1}$ by the last b bits.
4: **Merging**: Form pairs of columns in each partition to obtain $\mathbf{G}_i$
5: **end for**

Solving phase:

6: **for** $\mathbf{x} \in \{0,1\}^{k-bt}$ **do**
7: **return** the vector $\mathbf{x}$ that $[1\ \mathbf{x}]\mathbf{G}_t$ has minimal weight.
8: **end for**

Algorithm 2 Reduction of LF1

1: Partition $\mathbf{G}_{i-1} = V_0 \cup V_1 \cup \cdots \cup V_{2^b-1}$ s.t. the columns in V_j have the same last b bits.
2: **for** each V_j **do**
3: Randomly choose $\mathbf{v}^* \in V_j$ as the representative.
For $\mathbf{v} \in V_j, \mathbf{v} \neq \mathbf{v}^*$, $\mathbf{G}_i = \mathbf{G}_i \cup (\mathbf{v} + \mathbf{v}^*)$, ignoring the last b entries of 0.
4: **end for**

Algorithm 3 Reduction of LF2

1: Partition $\mathbf{G}_{i-1} = V_0 \cup V_1 \cup \cdots \cup V_{2^b-1}$ s.t. columns in V_j have the same last b bits.
2: **for** each V_j **do**
3: For each pair $\mathbf{v}, \mathbf{v}' \in V_j, \mathbf{v} \neq \mathbf{v}'$, $\mathbf{G}_i = \mathbf{G}_i \cup (\mathbf{v} + \mathbf{v}')$, ignoring the last b entries of 0.
4: **end for**

There are two approaches, called LF1 and LF2 in [22] to fulfill the merging procedure, sharing the same sorting approach with different merging strategies, which is described in the Algorithm 2 and 3, respectively. It is easy to see that LF1 works on pairs with a representative in each partition, which is discarded at last; while LF2 works on any pair. For each iteration in the reduction phase, the noise level is squared, as $e_j^{(i)} = e_{j_1}^{(i-1)} + e_{j_2}^{(i-1)}$ with the superscript (i) being the iteration step. Assume the noises remain independent at each step, we have $\Pr[\sum_{j=1}^{2^t} e_j = 0] = \frac{1}{2}(1 + \epsilon^{2^t})$ by the piling-up lemma.

3 The Previous Algorithm Using Covering Codes

In this section, we present a brief review of the algorithm using covering codes in [15], described in the following Algorithm 4.

Alg. 4 contains five main steps: step 1 transforms the problem into systematic form by Gaussian elimination (Line 2); step 2 performs several BKW steps (Line 3-5); jumping to step 4, it uses a covering code to rearrange the samples

Algorithm 4 The Algorithm using covering codes [15]

Input: n queries $(\mathbf{g}, z)$s of the (k, η)-LPN instance, the parameters b, t, k_2, l, w_1, w_2.

1: **repeat**
2: Pick random column permutation π and perform Gaussian elimination on $\pi(\mathbf{G})$, resulting in $[\mathbf{I}\ \mathbf{L}_0]$;
3: **for** $i = 1$ to t **do**
4: Perform LF1 reduction phase on $\mathbf{L}_{i-1}$ resulting in $\mathbf{L}_i$.
5: **end for**
6: Pick a $[k_2, l]$ linear code and group the columns of $\mathbf{L}_t$ by the last k_2 bits accord ing to their nearest codewords.
7: Set $k_1 = k - tb - k_2$;
8: **for** $\mathbf{x}'_1 \in \{0,1\}^{k_1}$ with $wt(\mathbf{x}'_1) \leq w_1$ **do**
9: Update the observed samples.
10: Use FWHT to compute the numbers of 1s and 0s for each $\mathbf{y} \in \{0,1\}^l$, and pick the best candidate.
11: Perform hypothesis testing with a threshold.
12: **end for**
13: **until**: Acceptable hypothesis is found.

(Line 6); step 3 guesses partial secret and Step 5 uses the FWHT to find the best candidate under the guessing, moreover it performs hypothesis testing to determine whether to repeat the algorithm (Line 7-13). Now we take a closer look at each step respectively.

Step 1. Gaussian elimination. This step systematizes the problem, i.e., change the positions of the secret vector bits without changing the associated noise level [21]. Precisely, from $\mathbf{z} = \mathbf{xG} + \mathbf{e}$, apply a column permutation π to make the first k columns of $\mathbf{G}$ linearly independent. Then form the matrix $\mathbf{D}$ such that $\hat{\mathbf{G}} = \mathbf{DG} = [\mathbf{I}\ \hat{\mathbf{g}}_{k+1}^T\ \hat{\mathbf{g}}_{k+2}^T\ \cdots\ \hat{\mathbf{g}}_n^T]$. Let $\hat{\mathbf{z}} = \mathbf{z} + [z_1\ z_2\ \cdots\ z_k]\hat{\mathbf{G}}$, thus $\hat{\mathbf{z}} = \mathbf{xD}^{-1}\hat{\mathbf{G}} + \mathbf{e} + [z_1\ z_2\ \cdots\ z_k]\hat{\mathbf{G}} = (\mathbf{xD}^{-1} + [z_1\ z_2\ \cdots\ z_k])\hat{\mathbf{G}} + \mathbf{e}$, where $\hat{\mathbf{z}} = [\mathbf{0}\ \hat{z}_{k+1}\ \hat{z}_{k+2}\ \cdots\ \hat{z}_n]$. Let $\hat{\mathbf{x}} = \mathbf{xD}^{-1} + [z_1\ z_2\ \cdots\ z_k]$, then $\hat{\mathbf{z}} = \hat{\mathbf{x}}\hat{\mathbf{G}} + \mathbf{e}$. From the special form of the first k components of $\hat{\mathbf{G}}$ and $\hat{\mathbf{z}}$, it is clear that $\Pr[\hat{x}_i = 1] = \Pr[e_i = 1] = \eta$. The cost of this step is dominated by the computation of $\mathbf{DG}$, which was reduced to $C_1 = (n-k)ka$ bit operations through table look-up in [15], where a is some fixed value.

Step 2. Collision procedure. This is the BKW part with the sort-and-match technique to reduce the dependency on the information bits [5, 22]. From $\hat{\mathbf{G}} = [\mathbf{I}\ \mathbf{L}_0]$, we iteratively process t steps of the BKW reduction on $\mathbf{L}_0$, resulting in a sequence of matrices $\mathbf{L}_i, i = 1, 2, \ldots, t$. Each $\mathbf{L}_i$ has $n - k - i2^b$ columns when adopting the LF1[3] type that discards about 2^b samples at each step. One also needs to update $\hat{\mathbf{z}}$ in the same fashion. Let $m = n - k - t2^b$, this procedure ends with $\mathbf{z}' = \mathbf{x}'\mathbf{G}' + \mathbf{e}'$, where $\mathbf{G}' = [\mathbf{I}\ \mathbf{L}_t]$ and $\mathbf{z}' = [\mathbf{0}\ z'_1\ z'_2\ \cdots\ z'_m]$.

[3] With the corrected number of queries, the algorithm in [15] exceeds the security bound of 80-bit. In order to obtain a complexity smaller than 2^{80}, the LF2 reduction step is actually applied.

The secret vector is reduced to a dimension of $k' = k - tb$, and also remains $\Pr[x'_i = 1] = \eta$ for $1 \le i \le k'$. The noise vector $\mathbf{e}' = [e_1 \ \cdots \ e_{k'} \ e'_1 \ \cdots \ e'_m]$, where $e'_i = \sum_{j \in \tau_i, |\tau_i| \le 2^t} e_j$ and τ_i contains the positions added up to form the $(k'+i)$-th column. The bias for e'_i is ϵ^{2^t} accordingly, where $\epsilon = 1 - 2\eta$. The complexity of this step is dominated by $C_2 = \sum_{i=1}^{t}(k+1-ib)(n-k-i2^b)$.

Step 3. Partial secret guessing. Divide $\mathbf{x}'$ into $[\mathbf{x}'_1 \ \mathbf{x}'_2]$, accordingly divide $\mathbf{G}' = \begin{bmatrix} \mathbf{G}'_1 \\ \mathbf{G}'_2 \end{bmatrix}$, where $\mathbf{x}'_1$ is of length k_1 and $\mathbf{x}'_2$ is of length k_2 with $k' = k_1 + k_2$. This step guesses all vectors $\mathbf{x}'_1 \in \{0,1\}^{k_1}$ that $wt(\mathbf{x}'_1) \le w_1$, where $wt(\ )$ is the Hamming weight of vectors. The complexity of this step is determined by updating $\mathbf{z}'$ with $\mathbf{z}' + \mathbf{x}'_1\mathbf{G}'_1$, denoted by $C_3 = m\sum_{i=0}^{w_1}\binom{k_1}{i}i$. The problem becomes $\mathbf{z}' = \mathbf{x}'_2\mathbf{G}'_2 + \mathbf{e}'$.

Step 4. Covering-code. A linear covering code is used in this step to further decrease the dimension of the secret vector. Use a $[k_2, l]$ linear code $\mathcal{C}$ with covering radius $d_{\mathcal{C}}$ to rewrite any $\mathbf{g}'_i \in \mathbf{G}'_2$ as $\mathbf{g}'_i = \mathbf{c}_i + \tilde{\mathbf{e}}_i$, where $\mathbf{c}_i$ is the nearest codeword in $\mathcal{C}$ and $wt(\tilde{\mathbf{e}}_i) \le d_{\mathcal{C}}$. Let the systematic generator matrix and its parity-check matrix of $\mathcal{C}$ be $\mathbf{F}$ and $\mathbf{H}$, respectively. Then the syndrome decoding technique is applied to select the nearest codeword. The complexity is cost in calculating syndromes $\mathbf{H}\mathbf{g}'^T_i, i = 1, 2, \ldots, m$, which was recursively computed in [15], as $C_4 = (k_2 - l)(2m + 2^l)$. Thus, $z'_i = \mathbf{x}'_2\mathbf{c}^T_i + \mathbf{x}'_2\tilde{\mathbf{e}}^T_i + e'_i, i = 1, 2, \ldots, m$. But if we use a specific concatenated code, the complexity formula of the syndrome decoding step will differ, as we stated later.

In [15], $\epsilon' = (1 - 2\frac{d}{k_2})^{w_2}$ is used to determine the bias introduced by covering, where d is the expected distance bounded by the sphere-covering bound, i.e., d is the smallest integer that $\sum_{i=0}^{d}\binom{k_2}{i} > 2^{k_2 - l}$, and w_2 is an integer that bounds $wt(\mathbf{x}'_2)$. But, we find that it is not proper to consider the components of error vector $\tilde{\mathbf{e}}_i$ as independent variables, which is also pointed out in [6]. Then Bogos et. al. update the bias estimation as follows: when the code has the optimal covering radius, the bias of $\langle \mathbf{x}'_2, \tilde{\mathbf{e}}_i \rangle = 1$ assuming that $\mathbf{x}'_2$ has weight w_2 can be found according to

$$\Pr[\langle \mathbf{x}'_2, \tilde{\mathbf{e}}_i \rangle = 1 | wt(\mathbf{x}'_2) = w_2] = \frac{1}{S(k_2, d)} \sum_{i \le d,\ i\ odd} \binom{c}{i} S(k_2 - w_2, d - i)$$

where $S(k_2, d)$ is the number of k_2-bit strings with weight at most d. Then the bias is computed as $\delta = 1 - 2\Pr[\langle \mathbf{x}'_2, \tilde{\mathbf{e}}_i \rangle = 1 | wt(\mathbf{x}'_2) = w_2]$, and the final complexity is derived by dividing a factor of the sum of covering chunks.[4] Later based on the calculation of the bias in [6], the authors of [15] further require that the

[4] We feel that there are some problems in the bias estimation in Bogos et. al. paper. In their work, the bias is computed as $1 - 2\Pr[(x, e) = 1 \mid wt(x) = c]$ with the conditional probability other than the normal probability Pr[(x,e)=1]. Note that the latter can be derived from the total probability formula by traversing all the conditions. Further, the weights of several secret chunks are assumed in a way that facilitates the analysis, which need to be divided at last to assure its occurrence. Here instead of summing up these partial conditional probabilities, they should be

Hamming weight bound w_2 is the largest weight of $\mathbf{x}_2'$ that the bias $\tilde{\epsilon}(w_2)$ is not smaller than ϵ_{set}, where ϵ_{set} is a preset bias. Still this holds with probability.

Step 5. Subspace hypothesis testing. It is to count the number of equality $z_i' = \mathbf{x}_2'\mathbf{c}_i^T$ in this step. Since $\mathbf{c}_i = \mathbf{u}_i\mathbf{F}$, one can count the number of equality $z_i' = \mathbf{y}\mathbf{u}_i^T$ equivalently, for $\mathbf{y} = \mathbf{x}_2'\mathbf{F}^T$. Group the samples $(\mathbf{g}_i', z_i')$ in sets $\mathcal{L}(\mathbf{c}_i)$ according to the nearest codewords and define the function $f(\mathbf{c}_i) = \sum_{(\mathbf{g}_i', z_i') \in \mathcal{L}(\mathbf{c}_i)} (-1)^{z_i'}$ on the domain of $\mathcal{C}$. Due to the bijection between $\mathbb{F}_2^l$ and $\mathcal{C}$, define the function $g(\mathbf{u}) = f(\mathbf{c}_i)$ on the domain of $\mathbb{F}_2^l$, where $\mathbf{u}$ represents the first l bits of $\mathbf{c}_i$ for the systematic feature of $\mathbf{F}$. The Walsh transform of g is defined as $\{G(\mathbf{y})\}_{\mathbf{y} \in \mathbb{F}_2^l}$, where $G(\mathbf{y}) = \sum_{\mathbf{u} \in \mathbb{F}_2^l} g(\mathbf{u})(-1)^{\langle \mathbf{y}, \mathbf{u} \rangle}$. The authors considered the best candidate as $\mathbf{y}_0 = \arg\max_{\mathbf{y} \in \mathbb{F}_2^l} |G(\mathbf{y})|$. This step calls for the complexity $C_5 = l2^l \sum_{i=0}^{w_1} \binom{k_1}{i}$, which runs for every guess of $\mathbf{x}_1'$ using the FWHT [7]. Note that if some column can be decoded into several codewords, one needs to run this step more times.

Analysis. In [15], it is claimed that it calls for approximately $m \approx 1/(\epsilon^{2^{t+1}} \epsilon'^2)$ samples to distinguish the correct guess from the others, and estimated $n \approx m + k + t2^b$ as the initial queries needed when adopting LF1 in the process. We find that this is highly underestimated. Then they correct it as $4l\ln 2/\epsilon_f^2$ in the presentation at Asiacrypt 2014, and adopt LF2 reduction steps with about $3 \cdot 2^b$ initial queries.

Recall that two assumptions are made regarding to the Hamming weight of secret vector, and it holds with probability $\Pr(w_1, k_1) \cdot \Pr(w_2, k_2)$, where $\Pr(w, k) = \sum_{i=0}^{w} (1-\eta)^{k-i} \eta^i \binom{k}{i}$ since $\Pr[x_i' = 1] = \eta$. If any assumption is invalid, one needs to choose another permutation to run the algorithm again. The authors showed the number of bit operations required for a success run of the algorithm using covering codes as

$$C = \frac{C_1 + C_2 + C_3 + C_4 + C_5}{\Pr(w_1, k_1)\Pr(w_2, k_2)}.$$

4 Our Improvements and Analysis

In this section, we present the core improvement and optimizations of our new algorithm with complexity analysis.

4.1 Embedding Cascaded Perfect Codes

First note that in [15], the explicit code constructions for solving those LPN instances to support the claimed attacks[5] are not provided. Second, it is suspicious whether there will be a good estimation of the bias, with the assumption

multiplied together. The Asiacrypt'14 paper has the similar problem in their analysis. Our theoretical derivation is different and new. We compute $\Pr[(x, e) = 1]$ according to the total probability formula strictly and thus the resultant bias precisely without any assumption, traversing all the conditional probabilities $\Pr[(x, e) = 1 \mid wt(e) = i]$.

[5] There is just a group of particular parameters for $(512, 1/8)$-LPN instance given in their presentation at Asiacrypt 2014, but the other LPN instances are missing.

of Hamming weight restriction, which is crucial for the exact estimate of the complexity. Instead, here we provide a *generic* method to construct the covering codes explicitly and compute the bias accurately.

Covering code is a set of codewords in a space with the property that every element of the space is within a fixed distance to some codeword, while in particular, perfect code is a covering code of minimal size. Let us first look at the perfect codes.

Definition 2 (Perfect code [24]). *A code $\mathcal{C} \subset Q^n$ with a minimum distance $2e+1$ is called a perfect code if every $\mathbf{x} \in Q^n$ has distance $\leq e$ to exactly one codeword, where Q^n is the n-dimensional space.*

From this definition, there exists one and only one decoding codeword in the perfect code for each vector in the space.[6] It is well known that there exists only a limited kinds of the binary perfect codes, shown in Table 2. Here e is indeed

Table 2. Types of all binary perfect codes

e	n	l	*Type*
0	n	n	$\{0,1\}^n$
1	2^r-1	2^r-r-1	Hamming code
3	23	12	Golay code
e	$2e+1$	1	repetition code
e	e	0	$\{\mathbf{0}\}$

the covering radius $d_{\mathcal{C}}$.

Confined to finite types of binary perfect codes and given fixed parameters of $[k_2, l]$, now the challenge is to efficiently find the configuration of some perfect codes that maximize the bias. To solve this problem, we first divide the $\mathbf{G}_2'$ matrix into several chunks by rows partition, and then cover each chunk by a certain perfect code. Thereby each $\mathbf{g}_i' \in \mathbf{G}_2'$ can be uniquely decoded as $\mathbf{c}_i$ chunk by chunk. Precisely, divide $\mathbf{G}_2'$ into h sub-matrices as

$$\mathbf{G}_2' = \begin{bmatrix} \mathbf{G}_{2,1}' \\ \mathbf{G}_{2,2}' \\ \vdots \\ \mathbf{G}_{2,h}' \end{bmatrix}.$$

For each sub-matrix $\mathbf{G}_{2,j}'$, select a $[k_{2,j}, l_j]$ perfect code $\mathcal{C}_j$ with the covering radius $d_{\mathcal{C}_j}$ to regroup its columns, where $j = 1, 2, \ldots, h$. That is, $\mathbf{g}_{i,j}' = \mathbf{c}_{i,j} + \tilde{\mathbf{e}}_{i,j}$, $wt(\tilde{\mathbf{e}}_{i,j}) \leq d_{\mathcal{C}_j}$ for $\mathbf{g}_{i,j}' \in \mathbf{G}_{2,j}'$, where $\mathbf{c}_{i,j}$ is the only decoded codeword in

[6] There exists exactly one decodable code word for each vector in the space, which facilitates the definition of the basic function in the Walsh transform. For other codes, the covering sphere may be overlapped, which may complicate the bias/complexity analysis in an unexpected way. It is our future work to study this problem further.

$\mathcal{C}_j$. Then we have

$$
\begin{aligned}
z'_i = \mathbf{x}'_2 \mathbf{g}'^T_i + e'_i &= \sum_{j=1}^{h} \mathbf{x}'_{2,j} \mathbf{g}'^T_{i,j} + e'_i \\
&= \sum_{j=1}^{h} \mathbf{x}'_{2,j} (\mathbf{c}_{i,j} + \tilde{\mathbf{e}}_{i,j})^T + e'_i = \sum_{j=1}^{h} \mathbf{x}'_{2,j} \mathbf{c}^T_{i,j} + \sum_{j=1}^{h} \mathbf{x}'_{2,j} \tilde{\mathbf{e}}^T_{i,j} + e'_i,
\end{aligned}
$$

where $\mathbf{x}'_2 = [\mathbf{x}'_{2,1}, \mathbf{x}'_{2,2}, \ldots, \mathbf{x}'_{2,h}]$ is partitioned in the same fashion as that of $\mathbf{G}'_2$. Denote the systematic generator matrix of $\mathcal{C}_j$ by $\mathbf{F}_j$. Since $\mathbf{c}_{i,j} = \mathbf{u}_{i,j}\mathbf{F}_j$, we have

$$
\begin{aligned}
z'_i &= \sum_{j=1}^{h} \mathbf{x}'_{2,j} \mathbf{F}^T_j \mathbf{u}^T_{i,j} + \sum_{j=1}^{h} \mathbf{x}'_{2,j} \tilde{\mathbf{e}}^T_{i,j} + e'_i \\
&= [\mathbf{x}'_{2,1}\mathbf{F}^T_1, \mathbf{x}'_{2,2}\mathbf{F}^T_2, \ldots, \mathbf{x}'_{2,h}\mathbf{F}^T_h] \cdot \begin{bmatrix} \mathbf{u}^T_{i,1} \\ \mathbf{u}^T_{i,2} \\ \vdots \\ \mathbf{u}^T_{i,h} \end{bmatrix} + \sum_{j=1}^{h} \mathbf{x}'_{2,j} \tilde{\mathbf{e}}^T_{i,j} + e'_i.
\end{aligned}
$$

Let $\mathbf{y} = [\mathbf{x}'_{2,1}\mathbf{F}^T_1, \mathbf{x}'_{2,2}\mathbf{F}^T_2, \ldots, \mathbf{x}'_{2,h}\mathbf{F}^T_h]$, $\mathbf{u}_i = [\mathbf{u}_{i,1}, \mathbf{u}_{i,1}, \ldots, \mathbf{u}_{i,h}]$, and $\tilde{e}_i = \sum_{j=1}^{h} \tilde{e}_{i,j} = \sum_{j=1}^{h} \mathbf{x}'_{2,j} \tilde{\mathbf{e}}^T_{i,j}$. Then $z'_i = \mathbf{y}\mathbf{u}^T_i + \tilde{e}_i + e'_i$, which conforms to the procedure of Step 5. Actually, we can directly group $(\mathbf{g}'_i, z'_i)$ in the sets $\mathcal{L}(\mathbf{u})$ and define the function $g(\mathbf{u}) = \sum_{(\mathbf{g}'_i, z'_i) \in \mathcal{L}(\mathbf{u})} (-1)^{z'_i}$, for each $\mathbf{u}_i$ still can be read from $\mathbf{c}_i$ directly without other redundant bits due to the systematic feature of those generator matrices. According to this grouping method, each $(\mathbf{g}'_i, z'_i)$ belongs to only one set. Then we examine all the $\mathbf{y} \in \mathbb{F}^l_2$ by the Walsh transform $G(\mathbf{y}) = \sum_{\mathbf{u} \in \mathbb{F}^l_2} g(\mathbf{u})(-1)^{\langle \mathbf{y}, \mathbf{u} \rangle}$ and choose the best candidate.

Next, we consider the bias introduced by such a covering fashion. We find that it is reasonable to treat the error bits $\tilde{e}_{\cdot,j}$ coming from different perfect codes as independent variables, while the error components of $\tilde{\mathbf{e}}_{\cdot,j}$ within one perfect code will have correlations to each other (here we elide the first subscript i for simplicity). Thus, we need an algorithm to estimate the bias introduced by a single $[k, l]$ perfect code with the covering radius $d_{\mathcal{C}}$, denoted by $\text{bias}(k, l, d_{\mathcal{C}}, \eta)$.

Equivalently, it has to compute the probability $\Pr[\mathbf{x}\mathbf{e}^T = 1]$ at first, where $\Pr[x_i = 1] = \eta$. In order to ensure $\mathbf{x}\mathbf{e}^T = 1$, within the components equal to 1 in $\mathbf{e}$, there must be an odd number of corresponding components equal to 1 in $\mathbf{x}$, i.e., $|supp(\mathbf{x}) \cap supp(\mathbf{e})|$ is odd. Thereby for $wt(\mathbf{e}) = i, 0 \leq i \leq d_{\mathcal{C}}$, we have

$$
\Pr[\mathbf{x}\mathbf{e}^T = 1 | wt(\mathbf{e}) = i] = \sum_{\substack{1 \leq j \leq i \\ j \text{ is odd}}} \eta^j (1-\eta)^{i-j} \binom{i}{j}.
$$

Moreover, $\Pr[wt(\mathbf{e}) = i] = 2^l\binom{k}{i}/2^k$, as the covering spheres are disjoint for perfect codes. We have

$$\Pr[\mathbf{x}\mathbf{e}^T = 1] = \sum_{i=0}^{d_{\mathcal{C}}} \frac{2^l\binom{k}{i}}{2^k}\left(\sum_{\substack{1\le j\le i\\ j \text{ is odd}}} \eta^j(1-\eta)^{i-j}\binom{i}{j}\right).$$

Additionally,

$$\sum_{\substack{1\le j\le i\\ j \text{ is even}}} \eta^j(1-\eta)^{i-j}\binom{i}{j} + \sum_{\substack{1\le j\le i\\ j \text{ is odd}}} \eta^j(1-\eta)^{i-j}\binom{i}{j} = (\eta+1-\eta)^i,$$
$$\sum_{\substack{1\le j\le i\\ j \text{ is even}}} \eta^j(1-\eta)^{i-j}\binom{i}{j} - \sum_{\substack{1\le j\le i\\ j \text{ is odd}}} \eta^j(1-\eta)^{i-j}\binom{i}{j} = (1-\eta-\eta)^i,$$

we can simplify $\sum_{1\le j\le i,\ j \text{ is odd}} \eta^j(1-\eta)^{i-j}\binom{i}{j}$ as $[1-(1-2\eta)^i]/2$. Then we derive the bias introduced by embedding the cascading as $\tilde{\epsilon} = \prod_{j=1}^{h}\epsilon_j$ according to the pilling-up lemma, where $\epsilon_j = \text{bias}(k_{2,j}, l_j, d_{\mathcal{C}_j}, \eta) = 1 - 2\Pr[\mathbf{x}_{2,j}\tilde{\mathbf{e}}_{:,j}^T = 1]$ for $\mathcal{C}_j$. Note that this is an *accurate* estimation without any assumption on the hamming weights.

Now we turn to the task to search for the optimal cascaded perfect codes $\mathcal{C}_1$, $\mathcal{C}_2, \ldots, \mathcal{C}_h$ that will maximize the final bias, given a fixed $[k_2, l]$ pair according to the LPN instances.

Denote this process by an algorithm, called construction(k_2, l, η). First, we calculate the bias introduced by each type of perfect code exploiting the above algorithm bias$(k, l, d_{\mathcal{C}}, \eta)$. In particular, for Hamming code, we compute bias$(2^r - 1, 2^r - r - 1, 1, \eta)$ for $r : 2^r - 1 \le k_2$ and $2^r - r - 1 \le l$. For repetition code, we compute bias$(2r+1, 1, r, \eta)$ for $r : 2r+1 \le k_2$. We compute bias$(23, 12, 3, \eta)$ for the [23,12] Golay code, and always have bias$(n, n, 0, \eta)$ equal to 1 for $\{0,1\}^n, n = 1, 2, \ldots$. Also it can be proved that bias$(r, 0, r, \eta) = [\text{bias}(1, 0, 1, \eta)]^r$ for any r. Second, we transform the searching problem into an integer linear programming problem. Let the number of $[2^r - 1, 2^r - r - 1]$ Hamming code be x_r and the number of $[2r+1, 1]$ repetition code be y_r in the cascading. Also let the number of $[23, 12]$ Golay code, $[1, 0]$ code $\{0\}$ and $[1, 1]$ code $\{0, 1\}$ in the cascading be z, v and w respectively. Then the searching problem converts into the following form.

$$\begin{cases} \sum\limits_{\substack{r:2^r-1\le k_2\\ 2^r-r-1\le l}} (2^r-1)x_r + \sum\limits_{r:2r+1\le k_2} (2r+1)y_r + 23z + v + w = k_2, \\ \sum\limits_{\substack{r:2^r-1\le k_2\\ 2^r-r-1\le l}} (2^r-r-1)x_r + \sum\limits_{r:2r+1\le k_2} y_r + 12z + w = l, \end{cases}$$

$$\max \left(\prod_{\substack{r:2^r-1\le k_2 \\ 2^r-r-1\le l}} \text{bias}(2^r-1, 2^r-r-1, 1, \eta)^{x_r} \right) \cdot$$

$$\cdot \left(\prod_{r:2r+1\le k_2} \text{bias}(2r+1, 1, r, \eta)^{y_r} \right) \text{bias}(23, 12, 3, \eta)^z \text{bias}(1, 0, 1, \eta)^v.$$

We perform the logarithm operations on the target function and make it linear as

$$\max \sum_{\substack{r:2^r-1\le k_2 \\ 2^r-r-1\le l}} x_r \log[\text{bias}(2^r-1, 2^r-r-1, 1, \eta)] + v\log[\text{bias}(1, 0, 1, \eta)] +$$

$$+ \sum_{r:2r+1\le k_2} y_r \log[\text{bias}(2r+1, 1, r, \eta)] + z\log[\text{bias}(23, 12, 3, \eta)].$$

Given the concrete value of η, we can provide the optimal cascaded perfect

Table 3. Optimal cascaded perfect codes employed in Section 6.4

	LPN instances (k, η)	Parameters $[k_2, l]$	Cascaded perfect codes	h	$\log_2\tilde{\epsilon}$
LF1	(512, 1/8)	[172, 62]	$y_2 = 9$, $y_3 = 5$, $z = 4$	18	-15.1360
	(532, 1/8)	[182, 64]	$y_2 = 11, y_3 = 5$, $z = 4$	20	-16.3859
	(592, 1/8)	[207, 72]	$y_3 = 8$, $y_4 = 4$, $z = 5$	17	-18.8117
LF2	(512, 1/8)	[170, 62]	$y_2 = 10, y_3 = 4$, $z = 4$	18	-14.7978
	(532, 1/8)	[178, 64]	$y_2 = 13, y_3 = 3$, $z = 4$	20	-15.7096
	(592, 1/8)	[209, 72]	$y_3 = 7$, $y_4 = 5$, $z = 5$	17	-19.1578
LF(4)	(512, 1/8)	[174, 60]	$y_2 = 1$, $y_3 = 11, z = 4$	16	-15.9152
	(532, 1/8)	[180, 61]	$y_2 = 2$, $y_3 = 11, z = 4, v = 1$	18	-16.7328
	(592, 1/8)	[204, 68]	$y_2 = 14, y_3 = 6$, $z = 4$	24	-19.2240

codes with fixed parameters by Maple. We present in Table 3 the optimal cascaded perfect codes with the parameters chosen in Section 6.4 for our improved algorithm when adopting various BKW algorithms for different LPN instances. From this table, we can find that the optimal cascaded perfect codes usually select the $[23, 12]$ Golay code and the repetition codes with r at most 4.

It is worth noting that the above mentioned process is a *generic* method that can be applied to any (k, η)-LPN instance, and finds the optimal perfect codes combination to be embedded. In the current framework, the results are optimal in the sense that the concrete code construction/the bias is optimally derived from the integer linear programming.

4.2 Data Complexity Analysis

In this section, we present an analysis of the accurate number of queries needed for choosing the best candidate in details. We first point out the distinguisher

statistic S between the two distributions corresponding to the correct guess and the others. It is obvious to see that S obeys $Ber_{\frac{1}{2}(1-\epsilon_f)}$ if $\mathbf{y}$ is correct, thus we deduce $\Pr[S_i = 1] = \frac{1}{2}(1-\epsilon_f) = \Pr[z'_i \neq \mathbf{y}\mathbf{u}_i^T]$, where $\epsilon_f = \epsilon^{2^t}\epsilon'$ indicates the bias of the final noise for simplicity. Since $G(\mathbf{y})$ calculates the difference between the number of equalities and inequalities, we have $S = \sum_{i=1}^{m} S_i = \frac{1}{2}(m - G(\mathbf{y}))$. It is clear that the number of inequalities should be minimum if $\mathbf{y}$ is correct. Thus the best candidate is $\mathbf{y}_0 = \arg\min_{\mathbf{y}\in\mathbb{F}_2^l} S = \arg\max_{\mathbf{y}\in\mathbb{F}_2^l} G(\mathbf{y})$, rather than $\arg\max_{\mathbf{y}\in\mathbb{F}_2^l} |G(\mathbf{y})|$ claimed in [15]. Then, let $X_{A=B}$ be the indicator function of equality. Rewrite $S_i = X_{z'_i = \mathbf{y}\mathbf{u}_i^T}$ as usual. Then S_i is drawn from $Ber_{\frac{1}{2}(1+\epsilon_f)}$ if $\mathbf{y}$ is correct and $Ber_{\frac{1}{2}}$ if $\mathbf{y}$ is wrong, which is considered to be random. Take $S = \sum_{i=1}^{m} S_i$, we consider the ranking procedure for each possible $\mathbf{y} \in \mathbb{F}_2^l$ according to the decreasing order of the grade S_y.

Let $\mathbf{y}_r$ denote the correct guess, and $\mathbf{y}_w$ otherwise. Given the independency assumption and the central limit theorem, we have

$$\frac{S_{\mathbf{y}_r} - \frac{1}{2}(1+\epsilon_f)m}{\sqrt{\frac{1}{2}(1-\epsilon_f)\frac{1}{2}(1+\epsilon_f)m}} \sim \mathcal{N}(0,1), \text{ and } \frac{S_{\mathbf{y}_w} - \frac{1}{2}m}{\frac{1}{2}\sqrt{m}} \sim \mathcal{N}(0,1),$$

where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with the expectation μ and variance σ^2. Thus we can derive $S_{\mathbf{y}_r} \sim \mathcal{N}(\frac{1}{2}(1+\epsilon_f)m, \frac{1}{4}(1-\epsilon_f^2)m)$ and $S_{\mathbf{y}_w} \sim \mathcal{N}(\frac{m}{2}, \frac{m}{4})$. According to the additivity property of normal distributions, $S_{\mathbf{y}_r} - S_{\mathbf{y}_w} \sim \mathcal{N}(\frac{1}{2}\epsilon_f m, \frac{1}{4}(2-\epsilon_f^2)m)$. Therefore, we obtain the probability that a wrong $\mathbf{y}_w$ has a better rank than the right $\mathbf{y}_r$, i.e., $S_{\mathbf{y}_r} < S_{\mathbf{y}_w}$ is approximately $\Phi\left(-\sqrt{\epsilon_f^2 m/(2-\epsilon_f^2)}\right)$, where $\Phi(\cdot)$ is the distribution function of the standard normal distribution. Let $\rho = \epsilon_f^2 m/(2-\epsilon_f^2) \approx \frac{1}{2}\epsilon_f^2 m$, and this probability becomes $\Phi(-\sqrt{\rho}) \approx e^{-\rho/2}/\sqrt{2\pi}$. Since we just select the best candidate, i.e., $S_{\mathbf{y}_r}$ should rank the highest to be chosen. Thus $S_{\mathbf{y}_r}$ gets the highest grade with probability approximatively equal to $(1 - \Pr[S_{\mathbf{y}_r} < S_{\mathbf{y}_w}])^{2^l - 1} \approx \exp(-2^l e^{-\rho/2}/\sqrt{2\pi})$. It is necessary to have $2^l \leq e^{\rho/2}$, i.e., at least $m \geq 4l\ln 2/\epsilon_f^2$ to make the probability high. So far, we have derived the number of queries used by the authors of [15] in their presentation at Asiacrypt 2014.

Furthermore, we have made extensive experiments to check the real success probability according to different multiples of the queries. The simulations show that $m = 4l\ln 2/\epsilon_f^2$ provides a success probability of about 70%, while for $m = 8l\ln 2/\epsilon_f^2$ the success rate is closed to 1[7]. To be consistent with the practical experiments, we finally estimate m as $8l\ln 2/\epsilon_f^2$ hereafter. Updating the complexities for solving different LPN instances in [15] with the $m = 8l\ln 2/\epsilon_f^2$ number of queries, the results reveal that the algorithm in [15] is *not* so valid to break the 80-bit security bound. In addition, in [15] there is a regrettable missing of the concrete terminal condition. It was said that a false alarm can be recognized by hypothesis testing, without the derivation of the specific threshold. To

[7] It is also analyzed that it calls for a number of $8l\ln 2/\epsilon_f^2$ to bound the failure probability in [6]. However, it uses the Hoeffding inequality, which is the different analysis method from ours.

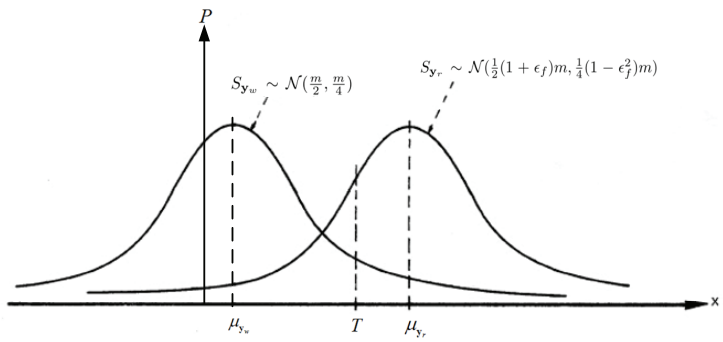

Fig. 1. Distributions according to $\mathbf{y}_w$ and $\mathbf{y}_r$

ensure the completeness of our improved algorithm, we solve this problem as follows. Denote the threshold by T, and adopt the selected best candidate $\mathbf{y}$ as correct if $S_{\mathbf{y}} \geq T$. The density functions of the corresponding distributions according to $\mathbf{y}_w$ and $\mathbf{y}_r$ are depicted in Fig. 1, respectively. Then it is clear to see that the probability for a false alarm is $P_f = \Pr[S_{\mathbf{y}} \geq T | \mathbf{y} \text{ is wrong}]$. It is easy to estimate P_f as $1 - \Phi(\lambda)$, where $\lambda = (T - \frac{m}{2})/\sqrt{\frac{m}{4}}$. Following our improvements described in Section 4.1, there is no assumption on the weight of $\mathbf{x}_2'$ now. The restricted condition is that the expected number of false alarms over all the $(2^l \sum_{i=0}^{w_1} \binom{k_1}{i}))/\Pr(w_1, k_1)$ basic tests is lower than 1. Thus we derive $\lambda = -\Phi^{-1}\left(\frac{\Pr(w_1,k_1)}{2^l \sum_{i=0}^{w_1} \binom{k_1}{i}}\right)$ and the algorithm terminal condition is $S_{\mathbf{y}} \geq T$, i.e., $G(\mathbf{y}) \geq \lambda\sqrt{m}$, for $S_{\mathbf{y}} = \frac{1}{2}(m + G(\mathbf{y}))$ according to the definition above.

4.3 An Vital Flaw in [15] That May Affect the Ultimate Results

As stated in [15], it took an optimized approach to calculate $\mathbf{D}\mathbf{g}^T$ for each column in $\mathbf{G}$. Concretely, for a fixed value s, divide the matrix $\mathbf{D}$ into $a = \lceil k/s \rceil$ parts, i.e., $\mathbf{D} = [\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_a]$, each sub-matrix containing s columns (possibly except the last one). Then store all the possible values of $\mathbf{D}_i\mathbf{x}^T$ for $\mathbf{x} \in \mathbb{F}_2^s$ in tables indexed by $i = 1, 2, \ldots, a$. For a vector $\mathbf{g} = [\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_a]$ partitioned according to $\mathbf{D}$, we have $\mathbf{D}\mathbf{g}^T = \mathbf{D}_1\mathbf{g}_1^T + \mathbf{D}_2\mathbf{g}_2^T + \cdots + \mathbf{D}_a\mathbf{g}_a^T$, where $\mathbf{D}_i\mathbf{g}_i^T$ can be read directly from the stored tables. The complexity of this step is to add those intermediate results together to derive the final result, shown as C_1 in Section 3. It was stated that the cost of constructing the tables is about $O(2^s)$, which can be negligible. Since the matrix $\mathbf{D}$ can only be obtained from the online querying and then refreshed for each iteration, this procedure should be reprocessed for each iteration and cannot be pre-computed in advance in the offline phase. Thus it is reasonable to include $PC_1/\left(\Pr(w_1, k_1)\Pr(w_2, k_2)\right)$ in the overall complexity.

5 Variants of the BKW Algorithm

In this section, we first present a theoretical analysis of the previous BKW algorithms with an emphasis on the differences in the reduction phase. Then we extend the heuristic algorithm LF2 into a series of variant algorithms denoted by LF(κ), as a basis of the improved algorithm proposed in Section 6. Furthermore, we verify the performance of these BKW algorithms in experiments.

5.1 LF1

LF1 works as follows. Choose a representative in each partition, add it to the rest of samples in the same partition and at last discard the representative, shown in the Algorithm 2 in Section 2.2. It is commonly believed that LF1 has no heuristics, and follows a rigorous analysis of its correctness and performance in theory. However, having checked the proof in [22], we find that the authors have overlooked the fact that the noise bits are no more independent after performing the XOR operations among the pairs of queries, which can be easily examined in the small instances. Thus there is no reason in theory to apply the pilling-up lemma for calculating the bias as shown in the proof. Thereby there is no need to treat it superior to other heuristic algorithms for the claimed strict proof.

Fortunately, by implementing LF1 algorithm, we find that the dependency does not affect the performance of the algorithm, shown in the Table 4. That is, the number of queries in theory with the independency assumption supports the corresponding success rate in practice. Thus we keep the independence assumption for the noise bits hereafter.

5.2 LF2

LF2 computes the sum of pairs from the same partition, shown in Algorithm 3 in Section 2.2. LF2 is more efficient and allows fewer queries compared to LF1. Let $n[i], i = 1, 2, \ldots, t$ be the excepted number of samples via the i-th BKW step, and $n[0] = n$. We impose a restriction that $n[i]$ is not larger than n for any i to LF2 with the following considerations. One is to control the dependence within certain limits, another is to preserve the number of samples not overgrowing, which will also stress on the complexity otherwise. The simulations done with the parameters under the restriction confirm the performance of LF2 shown in Table 4, and encounter with the statement that the operations of every pair have no visible effect on the success rate of the algorithm in [22].

5.3 Variants: LF(κ)

Here we propose a series of variants of the BKW algorithm, called LF(κ), which not only consider pairs of columns, but also consider κ-tuples that add to 0 in the last b entries. We describe the algorithm as follows. It is easy to see that the number of tuples satisfying the condition has an expectation of $E = \binom{n}{\kappa}2^{-b}$,

Algorithm 5 Reduction of LF(κ)

1: Find sufficient κ-tuples from $\mathbf{G}_{i-1}$ that add to 0 in the last b entries.
2: **for** each κ-tuple **do**
3: Calculate the sum of κ-tuple, joint it into $\mathbf{G}_i$ after discarding its last b bits of 0.
4: **end for**

given the birthday paradox. Similarly, define the excepted number of samples via the i-th BKW step as $n[i], i = 1, 2, \ldots, t$. We have $n[i] = \binom{n[i-1]}{\kappa} 2^{-b}$, which also applies to LF2 when $\kappa = 2$. We still impose the restriction that $n[i]$ is not larger than n for any i. The bias introduced by the variant decreases as ϵ^{κ^t}. We have implemented and run the variant algorithm for $\kappa = 3, 4$ under the data restriction, and verified the validity of these algorithms, shown in Table 4.

The extra cost of these variant BKW algorithms is to find a number of $n[i]$ such κ-tuples at each step. It is fortunate that this is the same as the κ-sum problem investigated in [25], which stated that the κ-sum problem for a single solution can be solved in $\kappa 2^{b/(1+\lfloor \log_2 \kappa \rfloor)}$ time and space [25]; moreover, one can find $n[i]$ solutions to the κ-sum problem with $n[i]^{1/(1+\lfloor \log_2 \kappa \rfloor)}$ times of the work for a single solution, as long as $n[i] \leq 2^{b/\lfloor \log_2 \kappa \rfloor}$ [25]. Thus this procedure of the variant BKW algorithms adds a complexity of $\kappa(2^b n[i])^{1/(1+\lfloor \log_2 \kappa \rfloor)}$ in time at each step, and $\kappa 2^{b/(1+\lfloor \log_2 \kappa \rfloor)}$ in space. Additionally, it stated that the lower bound of the computational complexity of κ-sum problem is $2^{b/\kappa}$. Thus it is possible to remove the limitation of the extra cost from the variant BKW algorithms if a better algorithm for κ-sum problem is proposed.

In Section 6, we present the results of the improved algorithms by embedding optimal cascaded perfect codes, which adopt LF1, LF2 and LF(4) at Step 2 respectively. We choose $\kappa = 4$ when adopting the variant BKW algorithm for the following reasons. If κ increases, the bias ϵ^{κ^t} introduced by LF(κ) falls sharply, and then we cannot find effective attack parameters. Since particularly stressed in [25] it takes a complexity of $2^{b/3}$ in time and space for a single solution when $\kappa = 4$, it calls for an extra cost of $(2^b n[i])^{1/3}$ in time at each step and $2^{b/3}$ in space when adopting the variant LF(4)[8] . Additionally, since the birthday paradox that $n[i] = (\binom{n[i-1]}{4}) \cdot 2^{-b}$, we derive that $n[i] = \frac{n[i-1]^4}{4!} \cdot 2^{-b}$, then $n[i-1] = \left(4! \cdot 2^b \cdot n[i]\right)^{1/4}$.

[8] Let us discuss whether LF(4) is equivalent to two consecutive LF2 steps. We can illustrate this in the aspect of the number of vectors remained. If we adopt LF(4) one step, then the number of vectors remained is about $s = \binom{n}{4} \cdot 2^{-b} = \frac{n^4}{4!} \cdot 2^{-b}$. While if we first to reduce the last b_1 positions with LF2, then the number of vectors remained is $t_1 = \binom{n}{2} \cdot 2^{-b_1} = \frac{n^2}{2} \cdot 2^{-b_1}$. Then, we do the second LF2 step regarding to the next $b - b_1$ positions and the number of vectors remained changes into $t_2 = \binom{t_1}{2} \cdot 2^{-(b-b_1)} = \frac{(n^2/2 \cdot 2^{-b_1})^2}{2} \cdot 2^{-(b-b_1)} = \frac{n^4}{8} \cdot 2^{-b-b_1}$. We can see that s is obviously not equal to t_2, so they are not equivalent. Indeed, LF(k) is algorithmically converted into several LF2, the point here is that we need to run the process a suitable number of times to find a sufficient number of samples.

5.4 Simulations

We have checked the performance as well as the correctness of the heuristic assumption by extensive simulations shown below. The experimental data adopt the numbers of queries estimated in theory, and obtain the corresponding success rate. In general, the simulations confirmed the validity of our theoretical analysis and the heuristic assumption, though the simulations are just conducted on some small versions of (k, η) for the limitation of computing power.

Table 4. Simulation data

LF1

Problem		Parameters			Results	
η	k	t	b	l	$\log_2 n$	success rate
0.01	60	5	10	10	12.38	98/100
0.02	65	5	12	5	14.35	92/100
0.05	60	4	12	12	14.16	96/100
0.10	70	4	15	10	17.62	88/100
0.15	40	3	10	10	15.27	99/100
0.20	50	3	14	8	18.66	92/100
0.30	65	2	20	25	22.14	84/100

LF2

Problem		Parameters			Results	
η	k	t	b	l	$\log_2 n$	success rate
0.01	60	5	9	15	9.95	96/100
0.02	50	5	9	5	9.96	89/100
0.05	55	4	11	11	11.93	99/100
0.10	70	4	16	6	16.90	91/100
0.15	55	3	15	10	16.75	97/100
0.20	60	3	18	6	19.76	90/100
0.30	60	2	18	24	19.66	80/100

LF(3)

Problem		Parameters			Results	
η	k	t	b	l	$\log_2 n$	success rate
0.01	60	3	14	18	8.30	98/100
0.05	80	3	25	5	13.76	94/100
0.10	55	2	22	11	12.85	90/100
0.15	60	2	27	6	15.34	93/100
0.20	40	1	20	20	12.07	90/100

LF(4)

Problem		Parameters			Results	
η	k	t	b	l	$\log_2 n$	success rate
0.02	65	2	22	21	8.87	95/100
0.05	70	2	29	12	11.38	92/100
0.15	50	1	29	21	12.14	90/100
0.20	45	1	32	13	13.16	91/100

5.5 A Technique to Reduce the Data Complexity

We briefly present a technique that applies to the last reduction step in LF2, aiming to reduce the number of queries. For simplicity, we still denote the samples at last step by $(\mathbf{g}_{t-1}, z)$, which accords with $z = \mathbf{x}\mathbf{g}_{t-1}^T + e$ and consists of k dimensions. Let the number of samples remained before running the last reduction step be n_{t-1} and the aim is to further reduce b dimensions at this step, i.e., $l = k - b$ for solving. This step runs as: (1) Sort the samples according to the last a bits of $\mathbf{g}_{t-1}$ $(a \leq b)$, and merge pairs from the same partition to generate the new samples $(\mathbf{g'}_{t-1}, z')$. (2) Sort $(\mathbf{g'}_{t-1}, z')$ into about $B = 2^{b-a}$ groups according to the middle $b - a$ bits of $\mathbf{g'}_{t-1}$ with each group containing about $m = \binom{n}{2}2^{-b}$ new samples. In each group, the value of $\mathbf{x}_{[l+1,l+a]}\mathbf{g}'^T_{[l+1,l+a]}$ is constant given an assumed value of $\mathbf{x}_{[l+1,l+a]}$, denoted by α. Here $\mathbf{x}_{[a,b]}$ means the interval of components within the vector $\mathbf{x}$. Thus in each group, the bias (denoted by ϵ) for the approximation of $z' = \mathbf{x}_{[1,l]}\mathbf{g}'^T_{[1,l]}$ contains the same sign,

which may be different from the real one, for $z' = \mathbf{x}_{[1,l]}\mathbf{g}'^T_{[1,l]} + \alpha + e'$ according to the assumed value of $\mathbf{x}_{[l+1,l+a]}$.

To eliminate the influence of the different signs among the different groups, i.e., no matter what value of $\mathbf{x}_{[l+1,l+a]}$ is assumed, we take χ^2-test to distinguish the correct $\mathbf{x}_{[1,l]}$. For each group, let the number of equalities be m_0. Then we have $S_i = (m_0 - \frac{m}{2})^2/(\frac{m}{2}) + (m - m_0 - \frac{m}{2})^2/(\frac{m}{2}) = W^2/m$, to estimate the distance between the distributions according to the correct $\mathbf{x}_{[1,l]}$ and wrong candidates considered as random, where W is the value of the Walsh transform for each $\mathbf{x}_{[1,l]}$. If $\mathbf{x}_{[1,l]}$ is correct, then $S_i \sim \chi_1^2(m\epsilon^2)$; otherwise, $S_i \sim \chi_1^2$, referred to [9]. Here, χ_M^2 denotes the χ^2-distribution with M degrees of freedom, whose mean is M and the variance is $2M$; and $\chi_M^2(\xi)$ is the non-central χ^2-distribution with the mean of $\xi + M$ and variance of $2(2\xi + M)$. Moreover, If $M > 30$, we have the approximation $\chi_M^2(\xi) \sim \mathcal{N}(\xi + M, 2(2\xi + M))$ [8]. We assume that the S_is are independent, and have the statistic $S = \sum_{j=1}^{B} S_i$. If $\mathbf{x}_{[1,l]}$ is correct, then $S \sim \chi_B^2(Bm\epsilon^2) \sim \mathcal{N}(B(1+\epsilon^2 m), 2B(1+2\epsilon^2 m)$; otherwise, $S \sim \chi_B^2 \sim \mathcal{N}(B, 2B)$, when $B > 30$, according to the additivity property of χ^2-distribution. Hereto, we exploit a similar analysis as Section 4.1 to estimate the queries needed to make the correct $\mathbf{x}_{[1,l]}$ rank highest according to the grade S, and result in $m \geq \frac{4l\ln 2}{B\epsilon^2}\left(1 + \sqrt{1 + \frac{B}{2l\ln 2}}\right)$ by solving a quadratic equation in m. Further amplify the result as $m > \frac{4l\ln 2}{\sqrt{B}\epsilon^2}$ and we find that this number decreases to a $\sqrt{B}$-th of the original one with the success rate close to 1. We have simulated this procedure for the independence assumption, and the experimental data verified the process. This technique allows us to overcome the lack of the data queries, while at the expense of some increase of the time complexity, thus we do not adopt it in the following Section 6.

6 Faster Algorithm with the Embedded Perfect Codes

In this section, we develop the improved algorithm for solving LPN with the key ideas introduced in the above Sections 4 and 5. Our improved algorithm for solving LPN has a similar framework as that in [15], but exploits a precise embedding of cascaded concrete perfect codes at Step 4 and optimizes each step with a series of techniques. The notations here follow those in Section 3.

We first formalize our improved algorithm in high level according to the framework, shown in Algorithm 6. We start by including an additional step to select the favorable queries.

Step 0. Sample selection. We select the data queries in order to transfer the (k, η)-LPN problem into a $(k - c, \eta)$-LPN problem compulsively. It works as follows. Just take the samples that the last c entries of $\mathbf{g}$ all equal 0 from the initial queries. Thus we reduce the dimension of the secret vector $\mathbf{x}$ by c-bit accordingly, while the number of initial queries needed may increase by a factor of 2^c. We only store these selected z and $\mathbf{g}$ without the last c bits of 0 to form the new queries of the reduced $(k - c, \eta)$-LPN instance, and for simplicity still

Algorithm 6 Improved Algorithm with the embedding cascaded perfect codes

Input: (k, η)-LPN instance of N queries, algorithm parameters $c, s, u, t, b, f, l, k_2, w_1$.

Select n samples that the last c bits of $\mathbf{g}$ all equal 0 from the initial queries, and store those selected z and $\mathbf{g}$ without the last c bits of 0.
Run the algorithm construction(k_2, l, η) to generate the optimal cascaded perfect codes and deduce the bias $\tilde{\epsilon}$ introduced by this embedding.
repeat
  Pick a random column permutation π and perform Gaussian elimination on $\pi(\mathbf{G})$ to derive $[\mathbf{I}\ \mathbf{L}_0]$;
  for $i = 1$ to t **do**
    Perform the BKW reduction step (LF1, LF2, LF(4)) on $\mathbf{L}_{i-1}$ resulting in $\mathbf{L}_i$.
  end for
  Based on the cascading, group the columns of $\mathbf{L}_t$ by the last k_2 bits according to their nearest codewords chunk by chunk.
  Set $k_1 = k - c - tb - k_2$;
  for $\mathbf{x}'_1 \in \{0,1\}^{k_1}$ with $wt(\mathbf{x}'_1) \leq w_1$ **do**
    Update the observed samples.
    Use FWHT to compute the numbers of 1s and 0s for each $\mathbf{y} \in \{0,1\}^l$, and pick the best candidate.
    Perform hypothesis testing with a threshold.
  end for
until: Acceptable hypothesis is found.

denoted by $(\mathbf{g}, z)$. Note that the parameter k used hereafter is actually $k - c$, and we do not substitute it for a clear comparison with [15].

As just stated, the number of initial queries is $N = 2^c n$. To search for the desirable samples that the last c entries of $\mathbf{g}$ being 0, we can just search for the samples that the Hamming weight of its last c entries equals to 0 with a complexity of $\log_2 c$ [17]. But, this can be sped up by pre-computing a small table of Hamming weight, and look up the table within a unit constant time $O(1)$. The pre-computation time and space can be ignored compared to those of the other procedures, for usually c is quite small. Thus, the complexity of this step is about $C_0 = N$.

Step 1. Gaussian elimination. This step is still to change the position distribution of the coordinates of the secret vector and is similar as that described in Section 3. Here we present several improvements of the dominant calculation of the matrix multiplication $\mathbf{DG}$.

In [15], this multiplication is optimized by table look-up as $\mathbf{D}\mathbf{g}^T = \mathbf{D}_1\mathbf{g}_1^T + \mathbf{D}_2\mathbf{g}_2^T + \cdots + \mathbf{D}_a\mathbf{g}_a^T$, now we further improve the procedure of constructing each table $\mathbf{D}_i\mathbf{x}^T$ for $\mathbf{x} \in \mathbb{F}_2^s$ by computing the products in a certain order. It is easy to see that the product $\mathbf{D}_i\mathbf{x}^T$ is a linear combination of some columns in $\mathbf{D}_i$. We partition $\mathbf{x} \in \mathbb{F}_2^s$ according to its weight. For $\mathbf{x}$ with $wt(\mathbf{x}) = 0$, $\mathbf{D}_i\mathbf{x}^T = 0$. For all $\mathbf{x}$ with $wt(\mathbf{x}) = 1$, we can directly read the corresponding column. Then for any $\mathbf{x}$ with $wt(\mathbf{x}) = 2$, there must be a $\mathbf{x}'$ with $wt(\mathbf{x}) = 1$ such that $wt(\mathbf{x} + \mathbf{x}') = 1$. Namely, we just need to add one column to the already obtained product $\mathbf{D}_i\mathbf{x}'^T$,

which is within k bit operations. Inductively, for any $\mathbf{x}$ with $wt(\mathbf{x}) = w$, there must be a $\mathbf{x}'$ with $wt(\mathbf{x}') = w - 1$ such that $wt(\mathbf{x} + \mathbf{x}') = 1$. Thus, the cost of constructing one table $\mathbf{D}_i\mathbf{x}^T, \mathbf{x} \in \mathbb{F}_2^s$ can be reduced to $k\sum_{i=2}^{s}\binom{s}{i} = (2^s - s - 1)k$. The total complexity is $PC_{11} = (2^s - s - 1)ka$ for a tables, which is much lower than the original $k^2 2^s$. We also analyze the memory needed for storing the tables $[\mathbf{x}, \mathbf{D}_i\mathbf{x}^T]_{\mathbf{x}\in\mathbb{F}_2^s}, i = 1, 2, \ldots, a$, which is $M_{11} = 2^s(s + k)a$.

Next, we present an optimization of the second table look-up to sum up the a columns, each of which has k dimensions. Based on the direct reading from the above tables, i.e, $\mathbf{D}_1\mathbf{g}_1^T + \mathbf{D}_2\mathbf{g}_2^T + \cdots + \mathbf{D}_a\mathbf{g}_a^T$, this addition can be divided into $\lceil k/u \rceil$ additions of a u-dimensional columns, depicted in the following schematic. We store a table of all the possible additions of a u-dimensional vectors and

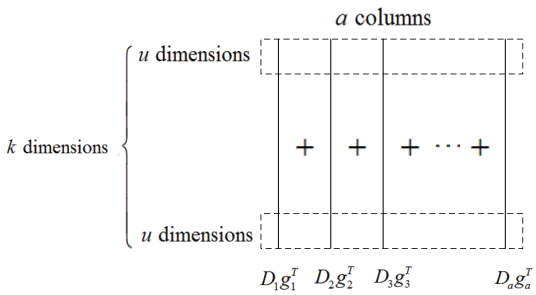

Fig. 2. Schematic of addition

read it $\lceil k/u \rceil$ times to compose the sum of a k-dimensional vectors required. Thus the complexity of Step 1 can be reduced to $C_1 = (n - k)(a + \lceil k/u \rceil)$ from $(n - k)ak$.

Now we consider the cost for constructing the tables of all the possible additions of a u-dimensional vectors. It will cost $2^{ua}u(a-1)$ bit operations by the simple exhaustive enumeration. We optimize this procedure as follows. It is true that any addition of a u-dimensional repeatable vectors can be reduced to the addition of less than or equal to a u-dimensional distinct nonzero vectors, for the sum of even number of same vectors equals $\mathbf{0}$. Thus, the problem transforms into the one that enumerates all the additions of i distinct nonzero u-dimensional vectors, where $i = 2, \ldots, a$. We find that every nonzero vector appears $\binom{2^u-1-1}{i-1}$ times in the enumeration of all the additions of the i distinct nonzero vectors. Then the total number of nonzero components of the vectors in the enumeration for i can be the upper bound of the bit operations for the addition, i.e., $\leq \binom{2^u-2}{i-1} \cdot \sum_{j=1}^{u}\binom{u}{j}j$ bit. Moreover, each nonzero vector appears the same number of times in the sums of the enumeration, which can be bounded by $\binom{2^u-1}{i}/(2^u - 1)$. We store the vectors as storing sparse matrix expertly and the memory required for this table is confined to $\left[\binom{2^u-2}{i-1} + \binom{2^u-1}{i}/(2^u - 1)\right] \cdot \left[\sum_{j=1}^{u}\binom{u}{j}j\right]$. We can simply $\sum_{j=1}^{u}\binom{u}{j}j$ as $\sum_{j=0}^{u-1}\binom{u-1}{j-1}u = u2^{u-1}$. Thus the total complexity for constructing this table 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. For each addition of a u-dimensional columns derived from the original a

k-dimensional columns, we discard all the even number of reduplicative columns and read the sum from the table. Moreover, this table can be pre-computed in the off-line phase and applied to each iteration. Since the table here is in the similar size to that used at Step 1 in [15], we still consider the complexity of the table look-up as $O(1)$, which is the same as that in [15]. This step has another improved method in [3].

Step 2. Collision procedure. This step still exploits the BKW reduction to make the length of the secret vector shorter. As stated in Section 5, we adopt the reduction mode of LF1, LF2 and LF(4) to this step, respectively. Similarly, denote the expected number of samples remained via the i-th BKW step by $n[i], i = 0, 1, \ldots, t$, where $n[0] = n, n[t] = m$ and m is the number of queries required for the final solving phase. First for LF1, $n[i] = n - k - i2^b$, as it discards about 2^b samples at each BKW step. We store a table of all the possible additions of two f-dimensional vectors similarly as that in Step 1. For the merging procedure at the i-th BKW step, we divide each pair of $(k+1-ib)$-dimensional columns into $\lceil \frac{k+1-ib}{f} \rceil$ parts, and read the sum of each part directly from the table. The cost for constructing the table is $PC_2 = f2^{f-1}(2^f - 2)$ and the memory to store the table is $M_2 = \frac{3}{2}f2^{f-1}(2^f - 2)$. Then the cost of this step is $C_2 = \sum_{i=1}^{t} \lceil \frac{k+1-ib}{f} \rceil (n - k - i2^b)$ for LF1, for the samples remained indicated the pairs found.

Second for LF2, we have $n[i] = \binom{n[i-1]}{2} 2^{-b}$ following the birthday paradox. We still do the merging as LF1 above and it calls for a cost of $C_2 = \sum_{i=1}^{t} \lceil \frac{k+1-ib}{f} \rceil n[i]$, 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.

Third for LF(4), we similarly have $n[i] = \binom{n[i-1]}{4} 2^{-b}$. We need to store a table of all the possible additions of four f-dimensional vectors. For the merging procedure at the i-th BKW step, we divide each 4-tuple of $(k + 1 - ib)$-dimensional columns into $\lceil \frac{k+1-ib}{f} \rceil$ parts, and read the sum of each part directly from the table. The cost of constructing the table is $PC_2 = \sum_{i=2}^{4} f2^{f-1} \binom{2^f-2}{i-1}$ and the memory to store the table is $\sum_{i=2}^{4} \frac{i+1}{i} f2^{f-1} \binom{2^f-2}{i-1}$. Additionally, there is still one more cost for finding the 4-tuples that add to 0 in the last b entries. This procedure has a cost of $(2^b n[i])^{1/3}$ as stated in Section 5.3. Hence, Step 2 has the complexity of $C_2 = \sum_{i=1}^{t} \left(\lceil \frac{k+1-ib}{f} \rceil n[i] + (2^b n[i])^{1/3} \right)$ for LF(4). Moreover, it needs another memory of $2^{b/3}$ to search for the tuples, i.e., $M_2 = \sum_{i=2}^{4} \frac{i+1}{i} f2^{f-1} \binom{2^f-2}{i-1} + 2^{b/3}$ for LF(4).

Step 3. Partial secret guessing. It still guesses all the vectors $\mathbf{x}'_1 \in \{0,1\}^{k_1}$ that $wt(\mathbf{x}'_1) \leq w_1$ and updates $\mathbf{z}'$ with $\mathbf{z}' + \mathbf{x}'_1 \mathbf{G}'_1$ at this step. We can optimize this step by the same technique used at Step 1 for multiplication. Concretely, the product $\mathbf{x}'_1 \mathbf{G}'_1$ is a linear combination of some rows in $\mathbf{G}'_1$. We calculate these linear combinations in the increasing order of $wt(\mathbf{x}'_1)$. For $wt(\mathbf{x}'_1) = 0$, $\mathbf{x}'_1 \mathbf{G}'_1 = 0$ and $\mathbf{z}'$ does not change. For all the $\mathbf{x}'_1$ with $wt(\mathbf{x}'_1) = 1$, we calculate the sum of $\mathbf{z}'$ and the corresponding row in $\mathbf{G}'_1$, which costs a complexity of m. Induc-

tively, for any $\mathbf{x}_1'$ with $wt(\mathbf{x}_1') = w$, there must be another $\mathbf{x}_1'$ of weight $w-1$ such that the weight of their sum equals 1, and the cost for calculating this sum based on the former result is m. Thus the cost of this step can be reduced from $m\sum_{i=0}^{w_1}\binom{k_1}{i}i$ to $C_3 = m\sum_{i=1}^{w_1}\binom{k_1}{i}$.

Step 4. Covering-coding. This step still works as covering each $\mathbf{g}_i' \in \mathbf{G}_2'$ with some fixed code to reduce the dimension of the secret vector. The difference is that we propose the explicit code constructions of the optimal cascaded perfect codes. According to the analysis in Section 4.1, we have already known the specific constructing method, the exact bias introduced by this constructing method and covering fashion, thus we do not repeat it here. One point to illustrate is how to optimize the process of selecting the codeword for each $\mathbf{g}_i' \in \mathbf{G}_2'$ online. From Table 3, we find that the perfect code of the longest length chosen for those LPN instances is the [23,12] Golay code. Thus we can pre-compute and store the nearest codewords for all the vectors in the space corresponding to each perfect code with a small complexity, and read it for each part of $\mathbf{g}_i'$ online directly. Here we take the [23,12] Golay code as an example, and the complexity for the other cascaded perfect codes can be ignored by taking into consideration their small scale of code length. Let $\mathbf{H}$ be the parity-check matrix of the $[23, 12]$ Golay code corresponding to its systematic generator matrix. We calculate all the syndromes $\mathbf{H}\mathbf{g}^T$ for $\mathbf{g} \in \mathbb{F}_2^{23}$ in a complexity of $PC_4 = 11 \cdot 23 \cdot 2^{23}$. Similarly, it can be further reduced by calculating them in an order of the increasing weight and also for the reason that the last 11 columns of $\mathbf{H}$ construct an identity matrix. Based on the syndrome decoding table of the $[23, 12]$ Golay code, we find the corresponding error vector $\mathbf{e}$ to $\mathbf{H}\mathbf{g}^T$ since $\mathbf{H}\mathbf{g}^T = \mathbf{H}(\mathbf{c}^T + \mathbf{e}^T) = \mathbf{H}\mathbf{e}^T$, and derive the codeword $\mathbf{c} = \mathbf{g} + \mathbf{e}$. We also obtain $\mathbf{u}$, which is the first 12 bits of $\mathbf{c}$. We store the pairs of $(\mathbf{g}, \mathbf{u})$ in the table and it has a cost of $M_4 = (23+12)\cdot 2^{23}$ in space. For the online phase, we just need to read from the pre-computed tables chunk by chunk, and the complexity of this step is reduced to $C_4 = mh$.

Step 5. Subspace hypothesis testing. At this step, it still follows the solving phase but with a little difference according to the analysis in Section 4.1. The complexity for this step is still $C_5 = l2^l\sum_{i=0}^{w_1}\binom{k_1}{i}$. Here we have to point that the best candidate chosen is $\mathbf{y}_0 = \arg\max_{\mathbf{y}\in\mathbb{F}_2^l} G(\mathbf{y})$, rather than $\arg\max_{\mathbf{y}\in\mathbb{F}_2^l} |G(\mathbf{y})|$. According to concrete data shown in the following tables, we estimate the success probability as $1 - \Pr[S_{\mathbf{y}} < T | \mathbf{y} \text{ is correct}]$ considering the missing events, and the results will be close to 1.

6.1 Complexity Analysis

First, we consider the final bias of the approximation $z_i' = \mathbf{y}\mathbf{u}_i^T$ in our improved algorithm. As the analysis in Section 4.1, we derive the bias introduced by embedding optimal cascaded perfect codes, denoted by $\tilde{\epsilon}$. The bias introduced by the reduction of the BKW steps is ϵ^{2^t} for adopting LF1 or LF2 at Step 2, while ϵ^{4^t} for adopting LF(4). Thus the final bias is $\epsilon_f = \epsilon^{2^t}\tilde{\epsilon}$ for adopting LF1 or LF2, and $\epsilon_f = \epsilon^{4^t}\tilde{\epsilon}$ for adopting LF(4).

Second, we estimate the number of queries needed. As stated in Section 4.2, it needs $m = 8l\ln 2/\epsilon_f^2$ queries to distinguish the correct guess from the others in the final solving phase. Then the number of queries for adopting LF1 at Step 2 is $n = m + k + t2^b$. For adopting LF2, the number of queries is computed as follows, $n[i] \approx \lceil (2^{b+1}n[i+1])^{1/2} \rceil, i = t-1, t-2, \ldots, 0$, where $n[t] = m$ and $n = n[0]$. Similarly for adopting LF(4), the number of queries is computed as $n[i] \approx \lceil (4!2^b n[i+1])^{1/4} \rceil$. Note that the number of initial queries needed in our improved algorithm should be $N = n2^c$ for the selection at Step 0, whatever the reduction mode is adopted at Step 2.

Finally, we present the complexity of our improved algorithm. The tables for vector additions at Step 1 and Step 2 can be constructed offline, as there is no need of the online querying data. The table for decoding at Step 4 can be calculated offline as well. Then the complexity of pre-computation is $Pre = PC_{12} + PC_2 + PC_4$. The memory complexity is about $M = nk + M_{11} + M_{12} + M_2 + M_4$ for storing those tables and the queries data selected. There remains one assumption regarding to the Hamming weight of the secret vector $\mathbf{x}'_1$, and it holds with the probability $\Pr(w_1, k_1)$, where $\Pr(w, k) = \sum_{i=0}^{w}(1-\eta)^{k-i}\eta^i\binom{k}{i}$ for $\Pr[x'_i = 1] = \eta$. Similarly, we need to choose another permutation to run the algorithm again if the assumption is invalid. Thus we are expected to meet this assumption within $1/\Pr(w_1, k_1)$ times iterations. The complexity of each iteration is $PC_{11} + C_1 + C_2 + C_3 + C_4 + C_5$. Hence, the overall time complexity of our improved algorithm online is

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

6.2 Complexity Results

Now we present the complexity results for solving the three core LPN instances by our improved algorithm when adopting LF1, LF2 and LF(4) at Step 2, respectively.

Note that all of the three LPN instances aim to achieve a security level of 80-bit, and this is indeed the *first* time to distinctly break the first two instances. Although we do not break the third instance, the complexity of our algorithm is quite close to the security level and the remained security margin is quite thin. More significantly, our improved algorithm can provide security evaluations to *any* LPN instances with the proposed parameters, which may be a basis of some cryptosystems.

We briefly illustrate the following tables. Here for each algorithm adopting a different BKW reduction type, we provide a corresponding table respectively; each contains a sheet of parameters chosen in the order of appearance and a sheet of overall complexity of our improved algorithm (may a sheet of queries numbers via each BKW step as well). There can be several choices when choosing the parameters, and we make a tradeoff in the aspects of time, data and memory. From Table 6 and 7, we can see the parameters chosen strictly follow the restriction that $n[i] \leq n$ for $i = 1, 2, \ldots, t$ for LF2 and LF(4), as stated in Section

Table 5. The complexity for solving the three LPN instances by our improved algorithm when adopting LF1

LPN instance	Parameters									Selected data $\log_2 n$
	c	s	u	t	b	f	l	k_2	w_1	
(512, 1/8)	5	51	8	5	63	31	62	172	1	66.291
(532, 1/8)	5	53	8	5	65	32	64	182	1	68.584
(592, 1/8)	4	59	9	5	73	36	72	207	1	75.557

LPN instance	Complexities			
	Time $\log_2 C$	Initial data $\log_2 N$	Memory $\log_2 M$	Pre-computation $\log_2 Pre$
(512, 1/8)	75.897	71.291	75.281	66.164
(532, 1/8)	78.182	73.584	77.629	68.053
(592, 1/8)	84.715	79.557	84.764	76.391

5. We also present an attack adopting LF(4) to $(592, 1/8)$-instance without the covering method but directly solving. The parameters are $c = 15$, $s = 59$, $u = 8$, $t = 3$, $b = 178$, $f = 17$, $l = k_2 = 35$, $w_1 = 1$, and the queries data via each BKW step is $n = 2^{60.859}$, $n[1] = 2^{60.853}$, $n[2] = 2^{60.827}$, $n[3] = 2^{60.725}$. The overall complexity is $C = 2^{81.655}$ in time, $N = 2^{75.859}$ for initial data, $M = 2^{72.196}$ in memory and $Pre = 2^{68.540}$ for the pre-computation.

Table 6. The complexity for solving three LPN instances by our improved algorithm when adopting LF2

LPN instance	Parameters									Selected data $\log_2 n$
	c	s	u	t	b	f	l	k_2	w_1	
(512, 1/8)	5	51	8	5	64	31	62	170	1	64.987
(532, 1/8)	7	53	8	5	66	32	64	178	1	66.983
(592, 1/8)	4	59	9	5	73	36	72	209	1	73.985

LPN instance	Data via each BKW step				
	$\log_2 n[1]$	$\log_2 n[2]$	$\log_2 n[3]$	$\log_2 n[4]$	$\log_2 n[5]$
(512, 1/8)	64.974	64.948	64.896	64.792	64.583
(532, 1/8)	66.966	66.932	66.863	66.726	66.453
(592, 1/8)	73.970	73.940	73.880	73.759	73.519

LPN instance	Complexities			
	Time $\log_2 C$	Initial data $\log_2 N$	Memory $\log_2 M$	Pre-computation $\log_2 Pre$
(512, 1/8)	74.732	69.987	73.983	66.164
(532, 1/8)	76.902	73.983	76.028	68.053
(592, 1/8)	83.843	77.985	83.204	76.391

Table 7. The complexity for solving three LPN instances by our improved algorithm when adopting LF(4)

LPN instance	Parameters									Selected data	Data via each BKW step	
	c	s	u	t	b	f	l	k_2	w_1	$\log_2 n$	$\log_2 n[1]$	$\log_2 n[2]$
(512, 1/8)	10	47	8	2	156	16	60	174	1	53.526	53.519	53.490
(532, 1/8)	15	47	8	2	162	17	61	180	1	55.504	55.433	55.149
(592, 1/8)	18	53	8	2	177	17	68	204	1	60.513	60.468	60.288

LPN instance	Complexities			
	Time $\log_2 C$	Initial data $\log_2 N$	Memory $\log_2 M$	Pre-computation $\log_2 Pre$
(512, 1/8)	72.844	63.526	68.197	68.020
(532, 1/8)	74.709	70.504	69.528	69.231
(592, 1/8)	81.963	78.513	70.806	69.231

Remark 1. All the results above strictly obey that $m = 8l\ln 2/\epsilon_f^2$, which is much more appropriate for evaluating the success probability, rather than the $m = 4l\ln 2/\epsilon_f^2$ which is chosen by the authors of [15] in their presentation at Asiacrypt 2014. If we choose the data as theirs for comparison, our complexity can be reduced to around 2^{71}, which is about 2^8 time lower than that in [15].

6.3 Concrete Attacks

Now we briefly introduce these three key LPN instances and the protocols based on them. The first one with parameter of (512, 1/8) is widely accepted in various LPN-based cryptosystems, e.g., HB$^+$ [19], HB$^\#$ [12] and LPN-C [13]. The 80-bit security of HB$^+$ is directly based on that of (512, 1/8)-LPN instance. Thus we can yield an active attack to break HB$^+$ authentication protocol straight forwardly. HB$^\#$ and LPN-C are two cryptosystems with the similar structures for authentication and encryption. There exist an active attack on HB$^\#$ and a chosen-plaintext attack on LPN-C. The typical parameter settings of the columns number are 441 for HB$^\#$, and 80 (or 160) for LPN-C. These two cryptosystems both consist of two version: secure version as Random- HB$^\#$ and LPN-C, efficient version as Toeplitz- HB$^\#$ and LPN-C. For the particularity of Toeplitz Matrices, if we attack its first column successively, then the cost for determining the remaining vectors can be bounded by 2^{40}. Thus we break the 80-bit security of these efficient versions employing Toeplitz matrices, i.e., Toeplitz-HB$^\#$ and LPN-C. For the random matrix case, the most common method is to attack it column by column. Then the complexity becomes a columns number multiple of the complexity attacking one $(512, 1/8)$-LPN instance[9]. That is, it has a cost of $441 \times 2^{72.177} \approx 2^{80.962}$ to attack Random-HB$^\#$, which slightly exceeds the

[9] Here, we adjust the parameter of $(512, 1/8)$-LPN instance in Table 6 that c changes into 16. Then the complexity of time, initial data, memory and pre-computation are respectively $C = 2^{72.177}$, $N = 2^{69.526}$, $M = 2^{68.196}$ and $Pre = 2^{68.020}$.

security level, and may be improved by some advanced method when conducting different columns. Similarly, the 80-bit security of RANDOM- LPN-C can be broke with a complexity of at most $160 \times 2^{72.177} \approx 2^{79.499}$.

The second LPN instance with the increased length (532, 1/8) is adopted as the parameter of an irreducible Ring-LPN instance employed in Lapin to achieve 80-bit security [16]. Since the Ring-LPN problem is believed to be not harder than the standard LPN problem, the security level can be break easily. It is urgent and necessary to increase the size of the employed irreducible polynomial in Lapin for 80-bit security. The last LPN instance with $(592, 1/8)$ is a new design parameter recommended to use in the future. However, we do not suggest to use it, for the security margin between our attack complexity and the security level is too small.

7 Conclusions

In this paper, we have proposed faster algorithms for solving the LPN problem based on an optimal precise embedding of cascaded concrete perfect codes, in the similar framework to that in [15], but with more careful analysis and optimized procedures. We have also proposed variants of BKW algorithms using tuples for collision and a technique to reduce the requirement of queries. The results beat all the previous approaches, and present efficient attacks against the LPN instances suggested in various cryptographic primitives. Our new approach is *generic* and is the best known algorithm for solving the LPN problem so far, which is practical to provide concrete security evaluations to the LPN instances with any parameters in the future designs. It is our further work to study the problem how to cut down the limitation of candidates, and meanwhile employ other type of good codes, such as nearly perfect codes.

Acknowledgements. The authors would like to thank one of the anonymous reviewers for very helpful comments. This work is supported by the program of the National Natural Science Foundation of China (Grant No. 61572482), National Grand Fundamental Research 973 Programs of China (Grant No. 2013CB-338002 and 2013CB834203) and the program of the National Natural Science Foundation of China (Grant No. 61379142).

References

1. M. Albrecht, C. Cid, J. C. Faugère, R. Fitzpatrick, and L. Perret. On the complexity of the BKW algorithm on LWE. *Designs, Codes and Cryptography*, 74(2):325–354, 2015.
2. C. Berbain, H. Gilbert, and A. Maximov. Cryptanalysis of Grain. In M. Robshaw, editor, *Fast Software Encryption–FSE'2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 15–29. Springer Berlin Heidelberg, 2006.
3. D. Bernstein. Optimizing linear maps modulo 2. available at `http://binary.cr.yp.to/linearmod2-20090830.pdf`

4. D. Bernstein and T. Lange. Never trust a bunny. In J. H. Hoepman and I. Verbauwhede, editors, *Radio Frequency Identification. Security and Privacy Issues*, volume 7739 of *Lecture Notes in Computer Science*, pages 137–148. Springer Berlin Heidelberg, 2013.
5. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4): 506–519, 2003.
6. S. Bogos, F. Tramer, and S. Vaudenay. On solving LPN using BKW and variants. available at `https://eprint.iacr.org/2015/049.pdf`.
7. P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: An algorithmic point of view. In L. Knudsen, editor, *Advances in Cryptology–EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer Berlin Heidelberg, 2002.
8. H. Cramér. *Mathematical methods of statistics*, volume 9. Princeton university press, 1999.
9. F. Drost, W. Kallenberg, D. Moore, and J. Oosterhoff. Power approximations to multinomial tests of fit. *Journal of the American Statistical Association*, 84(405):130–141, 1989.
10. Y. Dodis, E. Kiltz, K. Pietrzak, and D. Wichs. Message authentication, revisited. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology–EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 355–374. Springer Berlin Heidelberg, 2012.
11. A. Duc and S. Vaudenay. Helen: A public-key cryptosystem based on the LPN and the decisional minimal distance problems. In A. Youssef, A. Nitaj, and A. Hassanien, editors, *Progress in Cryptology–AFRICACRYPT 2013*, volume 7918 of *Lecture Notes in Computer Science*, pages 107–126. Springer Berlin Heidelberg, 2013.
12. H. Gilbert, M. Robshaw, and Y. Seurin. $HB^{\#}$: Increasing the security and efficiency of HB^{+}. In N. Smart, editor, *Advances in Cryptology–EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 361–378. Springer Berlin Heidelberg, 2008.
13. H. Gilbert, M. Robshaw, and Y. Seurin. 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*, volume 5126 of *Lecture Notes in Computer Science*, pages 679–690. Springer Berlin Heidelberg, 2008.
14. H. Gilbert, M. Robshaw, and H. Sibert. Active attack against HB^{+}: a provably secure lightweight authentication protocol. *Electronics Letters*, 41(21):1169–1170, 2005.
15. Q. Guo, T. Johansson, and C. Löndahl. Solving LPN using covering codes. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology–ASIACRYPT 2014*, volume 8873 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2014.
16. S. Heyse, E. Kiltz, V. Lyubashevsky, C. Paar, and K. Pietrzak. Lapin: An efficient authentication protocol based on ring-LPN. In A. Canteaut, editor, *Fast Software Encryption–FSE'2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 346–365. Springer Berlin Heidelberg, 2012.
17. Helger Lipmaa and Shiho Moriai Efficient Algorithms for Computing the Differential Properties of Addition. In M. Matsui, editor, *Fast Software Encryption–FSE'2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer Berlin Heidelberg, 2002.

18. N. Hopper and M. Blum. Secure human identification protocols. In C. Boyd, editor, *Advances in Cryptology–ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer Berlin Heidelberg, 2001.
19. A. Juels and S. Weis. Authenticating pervasive devices with human protocols. In V. Shoup, editor, *Advances in Cryptology–CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer Berlin Heidelberg, 2005.
20. E. Kiltz, K. Pietrzak, D. Cash, A. Jain, and D. Venturi. Efficient authentication from hard learning problems. In K. Paterson, editor, *Advances in Cryptology–EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 7–26. Springer Berlin Heidelberg, 2011.
21. Paul Kirchner. Improved generalized birthday attack. available at `http://eprint.iacr.org/2011/377.pdf`.
22. R. Levieil and P. A. Fouque. An improved LPN algorithm. In R. De Prisco and M. Yung, editors, *Security and Cryptography for Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer Berlin Heidelberg, 2006.
23. Y. Lu and S. Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In M. Franklin, editor, *Advances in Cryptology–CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 407–425. Springer Berlin Heidelberg, 2004.
24. J. H. Van Lint. *Introduction to coding theory*, volume 86. Springer Science & Business Media, 1999.
25. D. Wagner. A generalized birthday problem. In M. Yung, editor, *Advances in Cryptology–CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–304. Springer Berlin Heidelberg, 2002.