J Cryptol (2020) 33:1–33
https://doi.org/10.1007/s00145-019-09338-8

Solving LPN Using Covering Codes*

Qian Guo
Department of Electrical and Information Technology, Lund University, Lund, Sweden
Department of Informatics, Selmer Center, University of Bergen, Bergen, Norway
qian.guo@eit.lth.se

Thomas Johansson · Carl Löndahl
Department of Electrical and Information Technology, Lund University, Lund, Sweden
thomas.johansson@eit.lth.se
carl@grocid.net

Communicated by Alon Rosen.

Received 23 October 2015 / Revised 1 October 2019
Online publication 15 October 2019

Abstract. We present a new algorithm for solving the LPN problem. The algorithm has a similar form as some previous methods, but includes a new key step that makes use of approximations of random words to a nearest codeword in a linear code. It outperforms previous methods for many parameter choices. In particular, we can now solve the $(512, \frac{1}{8})$ LPN instance with complexity less than 2^{80} operations in expectation, indicating that cryptographic schemes like HB variants and LPN-C should increase their parameter size for 80-bit security.

Keywords. LPN, BKW, Covering codes, LPN-C, HB, Lapin.

1. Introduction

In recent years of modern cryptography, much effort has been devoted to finding efficient and secure low-cost cryptographic primitives targeting applications in very constrained hardware environments (such as RFID tags and low-power devices). Many proposals rely on the hardness assumption of *Learning Parity with Noise* (LPN), a fundamental problem in learning theory, which recently has also gained a lot of attention within the cryptographic society. The LPN problem is well studied, and it is intimately related to the problem of decoding random linear codes, which is one of the most important problems in coding theory. Being a supposedly hard problem, the LPN problem is a

*This paper is an extended version of [16] (https://doi.org/10.1007/978-3-662-45611-8_1). This paper was solicited by the Editors-in-Chief as the best paper from ASIACRYPT 2014, based on the recommendation of the program committee good candidate for post-quantum cryptography, where other classically hard problems such as factoring and the discrete log problem fall short. The inherent properties of LPN also make it ideal for lightweight cryptography.

The LPN problem can be informally stated as follows. We have an LPN oracle denoted Π_{LPN} that returns pairs of the form $(\mathbf{g}, \langle \mathbf{x}, \mathbf{g} \rangle + e)$, where $\mathbf{x}$ is an unknown but fixed binary vector, $\mathbf{g}$ is a binary vector with the same length but sampled from a uniform distribution, e is from a Bernoulli distribution, and $\langle \mathbf{x}, \mathbf{g} \rangle$ denotes the scalar product of vectors $\mathbf{x}$ and $\mathbf{g}$. The (search) LPN problem is to find the secret vector $\mathbf{x}$ given a fixed number of samples (oracle queries) from Π_{LPN}.

The first time the LPN problem was employed in a cryptographic construction was in the Hopper-Blum (HB) identification protocol [21]. HB is a minimalistic protocol that is secure in a *passive* attack model. Aiming to secure the HB scheme also in an *active* attack model, Juels and Weis [22], and Katz and Shin [23] proposed a modified scheme. The modified scheme, which was given the name HB$^+$, extends HB with one extra round. It was later shown by Gilbert et al. [15] that the HB$^+$ protocol is vulnerable to active attacks, in particular *man-in-the-middle attacks*, where the adversary is allowed to intercept and attack an ongoing authentication session to learn the secret. Gilbert et al. [13] subsequently proposed a variant of the Hopper-Blum protocol called HB$^\#$. Apart from repairing the protocol, the constructors of HB$^\#$ introduced a more efficient key representation using a variant of LPN called Toeplitz-LPN.

Gilbert et al. [14] proposed a way to use LPN in encryption of messages, which resulted in the cryptosystem LPN-C. Kiltz et al. [24] and Dodis et al. [10] showed how to construct message authentication codes (MACs) using LPN. The existence of MACs allows one to construct identification schemes that are provably secure against active attacks. The most recent contribution to LPN-based constructions is a two-round identification protocol called Lapin, proposed by Heyse et al. [20], and an LPN-based encryption scheme called Helen, proposed by Duc and Vaudenay [11]. The Lapin protocol is based on an LPN variant called Ring-LPN, where the samples are elements of a polynomial ring.

The two major threats against LPN-based cryptographic constructions are generic algorithms that decode random linear codes (information-set decoding (ISD)) and variants of the BKW algorithm, originally proposed by Blum et al. [3]. Being the asymptotically most efficient[1] approach, the BKW algorithm employs an iterated collision procedure on the queries. In each iteration, colliding entries sum together to produce a new entry with smaller dependency on the information bits but with an increased noise level. Once the dependency from sufficiently many information bits is removed, the remaining are exhaustively searched to find the secret. Although the collision procedure is the main reason for the efficiency of the BKW algorithm, it leads to a requirement of an immense amount of queries compared to ISD. Notably, for some cases, e.g., when the noise is very low, ISD yields the most efficient attack.

Levieil and Fouque [29] proposed to use fast Walsh–Hadamard transform in the BKW algorithm when searching for the secret. In an unpublished paper, Kirchner [25] suggested to transform the problem into systematic form, where each information (key) bit then appears as an observed symbol, perturbed by noise. This requires the adversary to only exhaust the biased noise variables rather than the key bits. When the error rate is

[1]For a fixed error rate.

Table 1. Comparison of different algorithms for solving LPN with parameters $(512, \frac{1}{8})$.

Algorithm	Complexity ($\log_2$)		
	Queries	Time	Memory
Levieil–Fouque [29]	75.7	87.5	84.8
Bernstein–Lange [4]	69.6	85.8	78.6
New algorithm (LF1)	65.0	80.7	74.0
New algorithm (LF2)	63.6	79.7	72.6

low, the noise variable search space is very small and this technique decreases the attack complexity. Building on the work by Kirchner [25], Bernstein and Lange [4] showed that the ring structure of Ring-LPN can be exploited in matrix inversion, further reducing the complexity of attacks on for example Lapin. None of the known algorithms manage to break the 80 bit security of Lapin. Nor do they break the parameters proposed in [29], which were suggested as design parameters of LPN-C [14] for 80-bit security.

1.1. *Contribution*

In this paper, we propose a new algorithm for solving the LPN problem based on [4,25]. We employ a new technique that we call subspace distinguishing, which exploits coding theory to decrease the dimension of the secret. The trade-off is a small increase in the sample noise. Our novel algorithm performs favorably in comparison to the state-of-the-art algorithms and affects the security of HB variants, Lapin and LPN-C. As an example, we attack the common $(512, \frac{1}{8})$-instance of LPN and question its 80-bit security barrier. A comparison of complexity of different algorithms[2] is shown in Table 1.

Let us explain the main idea of the paper in an informal way. The BKW algorithm will in each step remove the influence of b secret bits by colliding subvectors, at the cost of increasing the noise. So we can model a single step as reducing an LPN problem of dimension n and bias ϵ to an LPN problem of dimension $n - b$ and bias ϵ^2. The new main idea is that one can remove more secret bits if we now collide subvectors (linear combinations of secret bits) that are close in Hamming distance, but not necessarily the same. This will leave a few secret bits in each expression, but as the secret bits are biased and they can be considered as an additional noise term. Such a step reduces an LPN problem of dimension n and bias ϵ to an LPN problem of dimension $n - B$, where B is much larger than b, and the new bias is a bit smaller than ϵ^2. It is shown that LPN solvers that perform this new approach in the last step get an improved performance.

1.2. *Subsequent Work*

After the submission of this paper, a number of papers have appeared that further refine and improve upon this work. We mention the work of [5,6,32], and [12].

[2]The Bernstein–Lange algorithm is originally proposed for Ring-LPN, and by a slight modification [4], one can also apply it to the LPN instances. This modified algorithm shares several beginning steps (i.e., the steps of Gaussian elimination and the collision procedure) with the new algorithm, so we use the same implementation of these steps when computing their complexity, for a fair comparison.

To be specific, Bogos, Tramèr, and Vaudenay [5] presented a unified framework to study the existing LPN algorithms and also a tight theoretical bound to analyze the data complexity by using the Hoeffding bounds. Later, Zhang et al. [32] proposed a new method to analyze the bias introduced by the concatenation of several perfect codes, where the bias average rather than the bias conditioned on certain keys is employed. Bogos and Vaudenay [6] further clarified the underlying heuristic approximation and generalized the average bias analysis. They considered concrete code construction using concatenations of perfect and quasi-perfect codes. Note that firstly we can treat searching for large decodable linear codes with good covering property as a pre-computation task, and secondly, the analysis using bias average could produce a lower complexity estimation, which has been verified in our experiments where our bias estimation conditioned on key patterns matches the experimental data but is slightly conservative. In a recent paper [12], the idea of combining BKW and ISD was further investigated by Esser, Kübler and May.

1.3. *Organization*

The organization of the paper is as follows. In Sect. 2, we give some preliminaries and introduce the LPN problem in detail. Moreover, in Sect. 3 we give a short description of the BKW algorithm. We briefly describe the general idea of our new attack in Sect. 4 and more formally in Sect. 5. In Sect. 6, we analyze its complexity. The results when the algorithm is applied on various LPN-based cryptosystems are given in Sect. 7, which is followed by a section showing the experimental results. In Sect. 9, we describe some aspects of the covering-coding technique. Section 10 concludes the paper.

2. The LPN Problem

We now give a more thorough description of the LPN problem. Let Ber_η be the Bernoulli distribution and let $X \sim \mathsf{Ber}_\eta$ be a random variable with alphabet $\mathcal{X} = \{0, 1\}$. Then, $\mathbf{Pr}[X = 1] = \eta$ and $\mathbf{Pr}[X = 0] = 1 - \mathbf{Pr}[X = 1] = 1 - \eta$. The bias ϵ of X is given from $\mathbf{Pr}[X = 0] = \frac{1}{2}(1 + \epsilon)$, i.e., $\epsilon = 1 - 2\eta$. Let k be a security parameter, and let $\mathbf{x}$ be a binary vector of length k. We define the Hamming weight of a vector $\mathbf{v}$ as the number of nonzero elements, denoted by $w_{\mathrm{H}}(\mathbf{v})$, and let $\mathcal{B}_2(n, w)$ denote the Hamming ball which contains all the elements in $\mathbb{F}_2^n$ whose Hamming weight is no larger than w.

Definition 1. (LPN *oracle*) An LPN oracle Π_{LPN} for an unknown vector $\mathbf{x} \in \{0, 1\}^k$ with $\eta \in (0, \frac{1}{2})$ returns pairs of the form

$$\left(\mathbf{g} \overset{\$}{\leftarrow} \{0, 1\}^k, \langle \mathbf{x}, \mathbf{g} \rangle + e\right),$$

where $e \leftarrow \mathsf{Ber}_\eta$. Here, $\langle \mathbf{x}, \mathbf{g} \rangle$ denotes the scalar product of vectors $\mathbf{x}$ and $\mathbf{g}$.

We also write $\langle \mathbf{x}, \mathbf{g} \rangle$ as $\mathbf{x} \cdot \mathbf{g}^{\mathrm{T}}$, where $\mathbf{g}^{\mathrm{T}}$ is the transpose of the row vector $\mathbf{g}$. We receive a number n of noisy versions of scalar products of $\mathbf{x}$ from the oracle Π_{LPN}, and our task is to recover $\mathbf{x}$.

Problem 1. (LPN) *Given an* LPN *oracle* Π_{LPN}, *the* (k, η)-LPN *problem consists of finding the vector* $\mathbf{x}$. *An algorithm* $\mathcal{A}_{\mathrm{LPN}}(T, n, \delta)$ *using time at most* T *with at most* n *oracles queries solves* (k, η)-LPN *if*

$$\mathbf{Pr}\left[\mathcal{A}_{\mathrm{LPN}}(T, n, \delta) = \mathbf{x} : \mathbf{x} \overset{\$}{\leftarrow} \{0, 1\}^k\right] \geq \delta.$$

Let $\mathbf{y}$ be a vector of length n, and let $y_i = \langle \mathbf{x}, \mathbf{g}_i \rangle$. For known random vectors $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_n$, we can easily reconstruct an unknown $\mathbf{x}$ from $\mathbf{y}$ using linear algebra. In the LPN problem, however, we receive instead noisy versions of $y_i, i = 1, 2, \ldots, n$. Writing the noise in position i as $e_i, i = 1, 2, \ldots, n$, we obtain

$$z_i = y_i + e_i = \langle \mathbf{x}, \mathbf{g}_i \rangle + e_i.$$

In matrix form, the same relation is written as $\mathbf{z} = \mathbf{x}\mathbf{G} + \mathbf{e}$, where

$$\mathbf{z} = \begin{pmatrix} z_1 \ z_2 \ \cdots \ z_n \end{pmatrix}, \quad \mathbf{e} = \begin{pmatrix} e_1 \ e_2 \ \cdots \ e_n \end{pmatrix},$$

and the matrix $\mathbf{G}$ is formed as

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1^{\mathrm{T}} \ \mathbf{g}_2^{\mathrm{T}} \ \cdots \ \mathbf{g}_n^{\mathrm{T}} \end{pmatrix}.$$

This shows that the LPN problem is simply a decoding problem, where $\mathbf{G}$ is a random $k \times n$ generator matrix, $\mathbf{x}$ is the information vector, and $\mathbf{z}$ is the received vector after transmission of a codeword on the binary symmetric channel with error probability η.

2.1. *Piling-Up Lemma*

We recall the piling-up lemma, which is frequently used in analysis of the LPN problem.

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

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

2.2. *Complexity Estimates*

The computational complexity of a given algorithm can be given in many different ways. First, we may choose between giving asymptotic expressions or giving more explicit complexity estimates. For example, the BKW algorithm for solving LPN in dimension n is sub-exponential.

In this paper, we are primarily interested in explicit complexity estimates and we will thus try to estimate the number of operations required by an algorithm. We follow a long tradition of counting the number of "simple" bit operations. This includes reading a bit in memory, and it also does not have restrictions in memory size. Clearly, this model does not match an estimation of number of clock cycles on some CPU. In general, we expect the number of clock cycles to be less, since some word-oriented instructions can perform many bit operations in a single instruction.

3. The BKW Algorithm

The BKW algorithm, as proposed by Blum et al. [3], is an algorithm that solves the LPN problem in sub-exponential time, requiring $2^{\mathcal{O}(k/\log k)}$ queries and time. To achieve this, the algorithm uses an iterative sort-and-match procedure on the columns of the query matrix $\mathbf{G}$, which iteratively reduces the dimension of $\mathbf{G}$.

1. **Reduction phase** Initially, one searches for all combinations of two columns in $\mathbf{G}$ that add to zero in the last b entries. Let

$$\mathcal{M} \overset{\text{def}}{=} \{k-b+1, k-b+2, \dots, k\} \tag{1}$$

and define a filtering function $\phi_{\mathcal{M}} : \mathbb{F}_2^k \to \mathbb{F}_2^b$. Assume that one finds two columns $\mathbf{g}_{i_0}^{\mathrm{T}}, \mathbf{g}_{i_1}^{\mathrm{T}}$ such that

$$\mathbf{g}_{i_0} + \mathbf{g}_{i_1} = (* * \cdots * \underbrace{0\ 0 \cdots 0}_{b \text{ symbols}}), \tag{2}$$

where $*$ means any value, i.e., they belong to the same *partition* (or equivalence class) and fulfill $\phi_{\mathcal{M}}(\mathbf{g}_{i_0}) = \phi_{\mathcal{M}}(\mathbf{g}_{i_1})$. Then, a new vector

$$\mathbf{g}_1^{(1)} = \mathbf{g}_{i_0} + \mathbf{g}_{i_1} \tag{3}$$

is computed. Let $y_1^{(1)} = \left\langle \mathbf{x}, \mathbf{g}_1^{(1)} \right\rangle$. An *observed symbol* is also formed, corresponding to this new column by forming

$$z_1^{(1)} = z_{i_0} + z_{i_1} = y_1^{(1)} + e_1^{(1)} = \left\langle \mathbf{x}, \mathbf{g}_1^{(1)} \right\rangle + e_1^{(1)}, \tag{4}$$

where now $e_1^{(1)} = e_{i_0} + e_{i_1}$. It can be verified that $\mathbf{Pr}\left[e_1^{(1)} = 0\right] = \frac{1}{2} \cdot (1 + \epsilon^2)$. The algorithm proceeds by adding the same element, say $\mathbf{g}_{i_0}$, to the other elements in the partition forming

$$z_2^{(1)} = z_{i_0} + z_{i_2} = y_2^{(1)} + e_2^{(1)} = \left\langle \mathbf{x}, \mathbf{g}_2^{(1)} \right\rangle + e_2^{(1)}, \tag{5}$$

and so forth. The resulting columns are stored in a matrix $\mathbf{G}_1$,

$$\mathbf{G}_1 = \left((\mathbf{g}_1^{(1)})^{\mathrm{T}} \ (\mathbf{g}_2^{(1)})^{\mathrm{T}} \ \dots \ (\mathbf{g}_{n-2^b}^{(1)})^{\mathrm{T}} \right). \tag{6}$$

If n is the number of columns in $\mathbf{G}$, then the number of columns in $\mathbf{G}_1$ will be $n-2^b$. Note that the last b entries of every column in $\mathbf{G}_1$ are all zero. In connection to this matrix, the vector of observed symbols is

$$\mathbf{z}_1 = \left(z_1^{(1)} \ z_2^{(1)} \ \cdots \ z_{n-2^b}^{(1)} \right), \tag{7}$$

where $\mathbf{Pr}\left[z_i^{(1)} = y_i^{(1)}\right] = \frac{1}{2} \cdot (1+\epsilon^2)$, for $1 \le i \le n-2^b$.
We now iterate the same (with a new ϕ function), picking one column and then adding it to another suitable column in $\mathbf{G}_i$ giving a sum with an additional b entries being zero, forming the columns of $\mathbf{G}_{i+1}$. Repeating the same procedure, an additional $t-1$ times will reduce the number of unknown variables to $k-b\cdot t$ in the remaining problem.
For each iteration, the noise level is squared. By the piling-up lemma (Lemma 1), we have that

$$\mathbf{Pr}\left[\sum_{j=1}^{2^t} e_i = 0\right] = \frac{1}{2} \cdot \left(1+\epsilon^{2^t}\right). \tag{8}$$

Hence, the bias decreases quickly to low levels as t increases. Therefore, we want to keep t as small as possible.

2. **Solving phase** In the final step, the BKW algorithm looks for a column vector in $\mathbf{G}_t$ such that only the first bit of the vector is nonzero. If the algorithm finds such a vector, then that sample constitutes a very noisy observation the first bit x_1 of $\mathbf{x}$. The algorithm stores the observation and repeats the reduction-phase procedure with new samples from the oracle, until sufficiently many observations of the secret bit x_1 have been obtained. Then, it uses a majority decision to determine x_1. The whole procedure is given in Algorithm 1.

3.1. *LF1 and LF2 Variants*

The BKW algorithm is a powerful theoretical construction and because the algorithm operates solely on independent samples, it is possible to provide rigorous analysis using probabilistic arguments without heuristic assumptions. However, the provability comes at a quite high expense—the algorithm discards a lot of samples that could be used in solving the problem. This was first pointed out by Levieil and Fouque in [29]. They suggested that all samples should be kept after the reduction and not only the ones having weight 1. Instead of determining the secret bit by bit using majority decision, the whole $k-t\cdot b$ bit secret may be determined using Walsh transformation. The authors suggested

Algorithm 1 BKW

```
Input: Algorithm parameters $b, t, n \in \mathbb{N}$.
repeat
    (Reduction phase) Query the oracle for $n$ queries of the form $(\mathbf{g}, z)$; Create a query matrix $\mathbf{G} = (\mathbf{g}_1^T\ \mathbf{g}_2^T \cdots \mathbf{g}_n^T)$ and an observed vector $\mathbf{z} = (z_1\ z_2\ \cdots\ z_n)$;
    for $i \in \{1, 2, \ldots, n\}$ do
        $\mathcal{S} \leftarrow \mathcal{S} \cup (\mathbf{g}_i, z_i)$,
    for $i \in \{1, 2, \ldots, t\}$ do
        Partition $\mathcal{S}$ according to $b \cdot i$ last bits;
        for each partition $\mathcal{P} \in \mathcal{S}$ do
            Pick a random $(\mathbf{g}', z') \in \mathcal{P}$ and remove it from $\mathcal{P}$; Replace all remaining elements $(\mathbf{g}, z) \in \mathcal{P}$ with $(\mathbf{g} + \mathbf{g}', z + z')$;
    (Solving phase) Find a column vector in $\mathbf{G}_t$ such that only its first bit is non-zero and the remaining positions are all-zero. Then, the observed value $z_i$ is also an observation of $x_1$;
until sufficiently many observations have been obtained ;
Determine the secret bit $x_1$ by majority decision;
return $x_1$
```

two methods: LF1 and LF2—the methods are essentially the same, but differ in how the columns to be merged are chosen in the reduction phase.[3]

– LF1 picks a column in each partition and then adds it to the remaining samples in the same partition (entries having the same last b entries). This is identical to how the described BKW operates in its merging steps.
  The number of samples is reduced by 2^b after each merge operation. Hence, after a series of t merges, the number of samples is about

$$r(t) = n - t \cdot 2^b. \tag{9}$$

  The algorithm uses fast Walsh–Hadamard transform to determine the remaining secret of dimension $k - t \cdot b$. Thus, no samples are discarded and the algorithm does, in contrast to BKW, not query the oracle a multiple number of times. Therefore, a factor 2^b is lost in terms of query complexity.
  The LF1 method was subsequently adopted by Bernstein and Lange in [4].
– The other method, LF2, computes all pairs within the same partition. It produces more samples at the cost of increased dependency, thereby gaining more efficiency in practice.
  Given that there are on average $\frac{n}{2^b}$ samples in one partition, we expect around

$$2^b \binom{n/2^b}{2} \tag{10}$$

[3]One critical assumption for LF1/LF2 variants is that the samples are independent after several reduction steps. This assumption has been verified in [29] and also during our experiments.

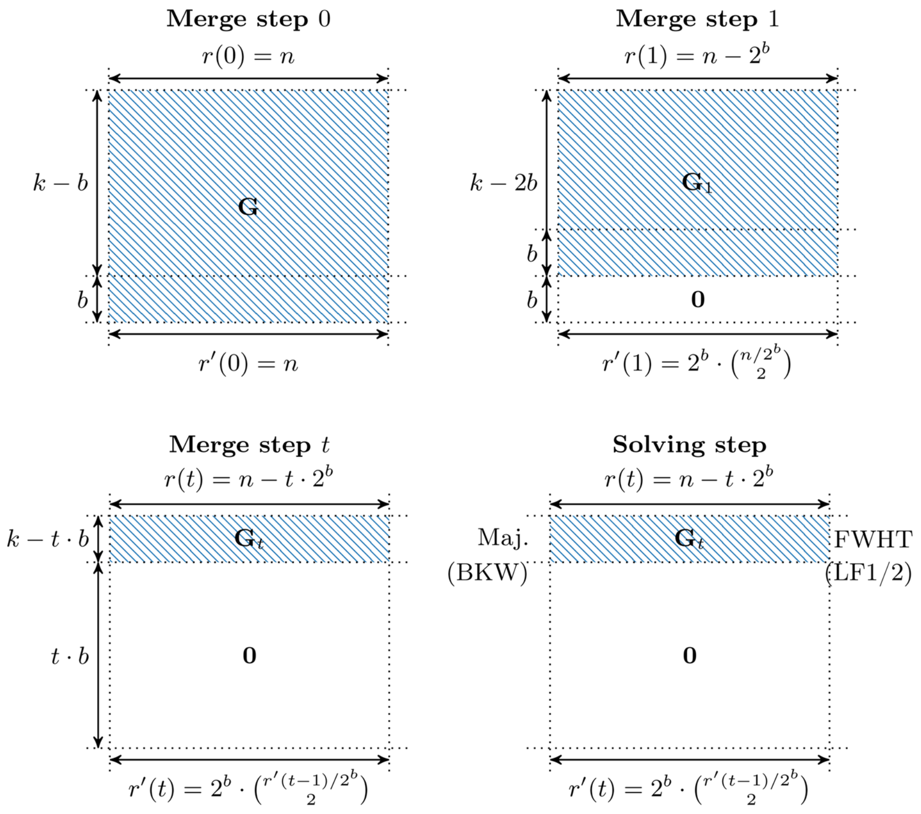

Fig. 1. In the above, we illustrate t merging steps and sample count at each t with respect to BKW/LF1, $r(t)$ and LF2, $r'(t)$.

possible samples at the end of one merge step in LF2, or more generally

$$r'(t) = 2^b \cdot \binom{r'(t-1)/2^b}{2}, \tag{11}$$

after t merging steps, with $r'(0) = n$. The number of samples preserves when setting $m = 3 \cdot 2^b$, and this setting is verified by an implementation in [29]. Like LF1, a fast Walsh–Hadamard transform (FWHT) is used to determine the secret. Combined with a more conservative use of samples, LF2 is expected to be at least as efficient as LF1 in practice. In particular, LF2 has great advantage when the attacker has restricted access to the oracle.

We have illustrated the different methods in Fig. 1.

4. Essential Idea

In this section, we try to give a very basic description of the idea used to give a new and more efficient algorithm for solving the LPN problem. A more detailed analysis will be provided in later sections.

Assume that we have an initial LPN problem described by

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_1^{\mathrm{T}} & \mathbf{g}_2^{\mathrm{T}} & \cdots & \mathbf{g}_n^{\mathrm{T}} \end{pmatrix}$$

and $\mathbf{z} = \mathbf{x}\mathbf{G} + \mathbf{e}$, where $\mathbf{z} = \begin{pmatrix} z_1 & z_2 & \cdots & z_n \end{pmatrix}$ and

$$z_i = y_i + e_i = \langle \mathbf{x}, \mathbf{g}_i \rangle + e_i.$$

As previously shown in [25] and [4], we may through Gaussian elimination transform $\mathbf{G}$ into systematic form. Assume that the first k columns are linearly independent and form the matrix $\mathbf{D}^{-1}$. With a change of variables $\hat{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1}$, we get an equivalent problem description with

$$\hat{\mathbf{G}} = \begin{pmatrix} \mathbf{I} & \hat{\mathbf{g}}_{k+1}^{\mathrm{T}} & \hat{\mathbf{g}}_{k+2}^{\mathrm{T}} & \cdots & \hat{\mathbf{g}}_n^{\mathrm{T}} \end{pmatrix}.$$

We compute

$$\hat{\mathbf{z}} = \mathbf{z} + \left(z_1, z_2, \ldots, z_k \right) \hat{\mathbf{G}} = \left(\mathbf{0}, \hat{z}_{k+1}, \hat{z}_{k+2}, \ldots, \hat{z}_n \right).$$

In this situation, one may start performing a number of BKW steps on columns $k+1$ to n, reducing the dimension k of the problem to something smaller. This will result in a new problem instance where noise in each position is larger, except for the first systematic positions. We may write the problem after performing t BKW steps in the form

$$\mathbf{G}' = \begin{pmatrix} \mathbf{I} & \mathbf{g}_1'^{\mathrm{T}} & \mathbf{g}_2'^{\mathrm{T}} & \cdots & \mathbf{g}_m'^{\mathrm{T}} \end{pmatrix}$$

and

$$\mathbf{z}' = \left(\mathbf{0}, z_1', z_2', \ldots z_m' \right),$$

where now $\mathbf{G}'$ has dimension $k' \times m$ with $k' = k - bt$ and m is the number of columns remaining after the t BKW steps. We have $\mathbf{z}' = \mathbf{x}'\mathbf{G}' + \mathbf{e}'$,

$$\mathbf{Pr}\left[x_i' = 0 \right] = \frac{1}{2}(1 + \epsilon)$$

and

$$\mathbf{Pr}\left[\mathbf{x}' \cdot \mathbf{g}_i'^{\mathrm{T}} = z_i \right] = \frac{1}{2}(1 + \epsilon^{2^t}).$$

Now we explain the basics of the new idea proposed in the paper. In a problem instance as above, we may look at the random variables $y'_i = \mathbf{x}' \cdot \mathbf{g}'^{\mathrm{T}}_i$. The bits in $\mathbf{x}'$ are mostly zero, but a few are set to one. Let us assume that c bits are set to one. Furthermore, $\mathbf{x}'$ is fixed for all i. We usually assume that $\mathbf{g}'_i$ is generated according to a uniform distribution. However, assuming that every column $\mathbf{g}'_i$ would be biased, i.e., every bit in a column position is zero with probability $1/2(1+\epsilon')$, we then observe that the variables y'_i will be biased, as

$$y'_i = \langle \mathbf{x}', \mathbf{g}'_i \rangle = \sum_{j=1}^{c} [\mathbf{g}'_i]_{k_j},$$

where $k_1, k_2, \ldots k_c$ are the bit positions where $\mathbf{x}'$ has value one (here $[\mathbf{x}]_y$ denotes bit y of vector $\mathbf{x}$). In fact, assuming that the variables $[\mathbf{g}'_i]_{k_j}$ are independently distributed,[4] variables y'_i will have bias $(\epsilon')^c$.

So how do we get the columns to be biased in the general case? We could simply hope for some of them to be biased, but if we need to use a larger number of columns, the bias would have to be small, giving a high complexity for an algorithm solving the problem. We propose instead to use a covering code to achieve something similar to what is described above. Vectors $\mathbf{g}'_i$ are of length k', so we consider a code of length k' and some dimension l. Let us assume that a generator matrix of this code is denoted $\mathbf{F}$. For each vector $\mathbf{g}'_i$, we now find the codeword in the code spanned by $\mathbf{F}$ that is closest (in Hamming sense) to $\mathbf{g}'_i$. Assume that this codeword is denoted $\mathbf{c}_i$. Then, we can write

$$\mathbf{g}'_i = \mathbf{c}_i + \mathbf{e}'_i,$$

where $\mathbf{e}'_i$ is a vector with biased bits. It remains to examine exactly how biased the bits in $\mathbf{e}'_i$ will be, but assume for the moment that the bias is ϵ'. Going back to our previous expressions, we can write

$$y'_i = \langle \mathbf{x}', \mathbf{g}'_i \rangle = \mathbf{x}' \cdot (\mathbf{c}_i + \mathbf{e}'_i)^{\mathrm{T}}$$

and since $\mathbf{c}_i = \mathbf{u}_i\mathbf{F}$ for some $\mathbf{u}_i$, we can write

$$y'_i = \mathbf{x}'\mathbf{F}^{\mathrm{T}} \cdot \mathbf{u}_i^{\mathrm{T}} + \mathbf{x}' \cdot \mathbf{e}'^{\mathrm{T}}_i.$$

We may introduce $\mathbf{v} = \mathbf{x}'\mathbf{F}^{\mathrm{T}}$ as a length l vector of unknown bits (linear combinations of bits from $\mathbf{x}'$) and again

$$y'_i = \mathbf{v} \cdot \mathbf{u}_i^{\mathrm{T}} + \mathbf{x}' \cdot \mathbf{e}'^{\mathrm{T}}_i.$$

[4]There are various approaches to estimate the bias introduced by coding. As the main goal in this section is to illustrate the gist of the new idea, we adopt the most straight-forward one, i.e., the one that assumes the variables each representing noise in a position in the error vector are independent. In a later section, when computing the algorithm complexity, a more accurate value is obtained by calculating the bias numerically (see Proposition 1).

Since we have $\mathbf{Pr}\left[y'_i = z'_i\right] = 1/2(1+\epsilon^{2^t})$, we get

$$\mathbf{Pr}\left[\mathbf{v}\cdot\mathbf{u}_i^{\mathrm{T}} = z'_i\right] = \frac{1}{2}(1+\epsilon^{2^t}(\epsilon')^c),$$

where ϵ' is the bias determined by the expected distance between $\mathbf{g}'_i$ and the closest codeword in the code we are using, and c is the number of positions in $\mathbf{x}'$ set to one. The last step in the new algorithm now selects about $m = \mathcal{O}\left(l/(\epsilon^{2^t}\cdot\epsilon'^c)^2\right)$ samples $z'_1, z'_2, \ldots, z'_m$ and for each guess of the 2^l possible values of $\mathbf{v}$, we compute how many times $\mathbf{v}\cdot\mathbf{u}_i^{\mathrm{T}} = z'_i$ when $i = 1, 2, \ldots, m$. As this step is similar to a correlation attack scenario, we know that it can be efficiently computed using fast Walsh–Hadamard transform. After recovering $\mathbf{v}$, it is an easy task to recover remaining unknown bits of $\mathbf{x}'$.

4.1. *A Toy Example*

In order to illustrate the ideas and convince the reader that the proposed algorithm can be more efficient than previously known methods, we consider an example. We assume an LPN instance of dimension $k = 160$, where we allow at most 2^{24} received samples and we allow at most around 2^{24} vectors of length 160 to be stored in memory. Furthermore, the error probability is $\eta = 0.1$.

For this particular case, we propose the following algorithm. Note that for an intuitive explanation, we assume the number of required samples to be $1/\epsilon_{tot}^2$, where ϵ_{tot} is the total bias. A rigorous complexity analysis of the new algorithm will be presented later.

1. The first step is to compute the systematic form,

$$\hat{\mathbf{G}} = \left(\mathbf{I}\ \hat{\mathbf{g}}_{k+1}^{\mathrm{T}}\ \hat{\mathbf{g}}_{k+2}^{\mathrm{T}}\ \cdots\ \hat{\mathbf{g}}_n^{\mathrm{T}}\right)$$

   and

$$\hat{\mathbf{z}} = \mathbf{z} + \left(z_1\ z_2\ \ldots\ z_k\right)\hat{\mathbf{G}} = \left(\mathbf{0}\ \hat{z}_{k+1}\ \hat{z}_{k+2}\ \ldots\ \hat{z}_n\right).$$

   Here, $\hat{\mathbf{G}}$ has dimension 160 and $\hat{\mathbf{z}}$ has length at most 2^{24}.
2. In the second step, we perform $t = 4$ merging steps (using the BKW/LF1 approach), the first step removing 22 bits and the remaining three each removing 21 bits. This results in $\mathbf{G}' = \left(\mathbf{I}\ \mathbf{g}_1'^{\mathrm{T}}\ \mathbf{g}_2'^{\mathrm{T}}\ \cdots\ \mathbf{g}_m'^{\mathrm{T}}\right)$ and $\mathbf{z}' = \left(\mathbf{0}\ z'_1\ z'_2\ \ldots\ z'_m\right)$, where now $\mathbf{G}'$ has dimension $75\times m$ and m is about $3\cdot 2^{21}$. We have $\mathbf{z}' = \mathbf{x}'\mathbf{G}'$,

$$\mathbf{Pr}\left[x'_i = 0\right] = \frac{1}{2}\cdot(1+\epsilon),$$

   where $\epsilon = 0.8$ and

$$\mathbf{Pr}\left[\langle\mathbf{x}', \mathbf{g}'_{\mathbf{i}}\rangle = z_i\right] = \frac{1}{2}\cdot(1+\epsilon^{16}).$$

   Hence, the resulting problem has dimension 75 and the bias is $\epsilon^{2^t} = (0.8)^{16}$.

3. In the third step, we then select a suitable code of length 75. In this example, we choose a block code which is a direct sum of 25 [3, 1, 3] repetition codes,[5] i.e., the dimension is 25. We map every vector $\mathbf{g}'_i$ to the nearest codeword by simply selecting chunks of three consecutive bits and replace them by either 000 or 111. With probability $\frac{3}{4}$, we will change one position and with probability $\frac{1}{4}$ we will not have to change any position. In total, we expect to change $(\frac{3}{4} \cdot 1 + \frac{1}{4} \cdot 0) \cdot 25$ positions. The expected weight of the length 75 vector $\mathbf{e}'_i$ is $\frac{1}{4} \cdot 75$, so the expected bias is $\epsilon' = \frac{1}{2}$. As $\mathbf{Pr}\left[x'_i = 1\right] = 0.1$, the expected number of nonzero positions in $\mathbf{x}'$ is 7.5. Assuming we have only $c = 6$ nonzero positions, we get

$$\mathbf{Pr}\left[\langle \mathbf{v}, \mathbf{u}_i \rangle = z'_i\right] = \frac{1}{2} \cdot \left(1 + 0.8^{16} \cdot \left(\frac{1}{2}\right)^6\right) = \frac{1}{2} \cdot (1 + 2^{-11.15}).$$

4. In the last step, we then run through 2^{25} values of $\mathbf{v}$ and for each of them we compute how often $\mathbf{v} \cdot \mathbf{u}_i^{\mathrm{T}} = z'_i$ for $i = 1, \ldots, 3 \cdot 2^{21}$. Again since we use fast Walsh–Hadamard transform, the cost of this step is not much more than 2^{25} operations.
5. The above four-step procedure forms one iteration of our solving algorithm, and we need to repeat it a few times. The expected number depends on the success probability of one iteration. For this particular repetition code, there are ≫bad events≪ that make the distinguisher fail. When two of the errors in $\mathbf{x}'$ fall into the same concatenation, then the bias is zero. If there are three errors in the same concatenation, then the bias is negative. To conclude, we can distinguish successfully if there are no more than 6 ones in $\mathbf{x}'$ and each of them falls into a distinct concatenation, i.e., the overall bias is at least $2^{-11.15}$. The successes probability[6] is thus

$$\sum_{i=0}^{6} \binom{25}{i} \cdot \binom{3}{1}^i \cdot \left(\frac{1}{10}\right)^i \cdot \left(\frac{9}{10}\right)^{75-i} \approx 0.28. \tag{12}$$

In comparison with other algorithms, the best approach we can find is the Kirchner [25] and the Bernstein and Lange [4] approaches, where one can do up to 5 merging steps. Removing 21 bits in each step leaves 55 remaining bits. Using fast Walsh–Hadamard transform with $0.8^{-64} = 2^{20.6}$ samples, we can include another 21 bits in this step, but there are still 34 remaining variables that needs to be guessed.

[5] In the sequel, we denote this code construction as concatenated repetition code. For this [75, 25, 3] linear code, the covering radius is 25, but we could see from this example that what matters is the average weight of the error vector, which is much smaller than 25.

[6] This explains why we need a more rigorous analysis. If we would assume that the noise variables in the error vector are independent, the success probability is about 0.37. This estimation is too optimistic, since if two of the errors in $\mathbf{x}'$ fall into the same code, the resulting zero bias totally ruin the statistical distinguishing procedure. We use a more accurate estimation in (12), which is further illustrated in Example 1.

Algorithm 2 New attacking algorithm in the LF1 setting

```
Input: Matrix $\mathbf{G}$ with $k$ rows and $n$ columns, received length $n$ vector $\mathbf{z}$ and algorithm parameters
$t, b, k'', l, w_0, \epsilon_{set}$

1  repeat
2      Pick a random column permutation $\pi$;
3      Perform Gaussian elimination on $\pi(\mathbf{G})$ resulting in $\hat{\mathbf{G}} = [\mathbf{I}|\mathbf{L}_0]$;
4      for $i = 1$ to $t$ do
5          Partition the columns of $\mathbf{L}_{i-1}$ by the last $b \cdot i$ bits;
6          Denote the set of columns in partition $s$ by $\mathcal{L}_s$;
7          Pick a vector $\mathbf{a}_{is} \in \mathcal{L}_s$;
8          for $(\mathbf{a} \in \mathcal{L}_s)$ and $(\mathbf{a} \neq \mathbf{a}_{is})$ do
9              $\mathbf{L}_i \leftarrow [\mathbf{L}_i | (\mathbf{a} + \mathbf{a}_{is})]$;
10     Pick a $[k'', l]$ linear code with good covering property;
11     Partition the columns of $\mathbf{L}_t$ by the middle non-all-zero $k''$ bits and group them by their
       nearest codewords;
12     Set $k_1 = k - tb - k''$;
13     for $\mathbf{x}'_2 \in \{0, 1\}^{k_1}$ with $wt(\mathbf{x}'_2) \leq w_0$ do
14         Update the observed samples;
15         for $\mathbf{v} \in \{0, 1\}^l$ do
16             Use Fast Walsh-Hadamard Transform to compute the numbers of 1s and 0s
               observed respectively;
17             Perform hypothesis testing;
18 until acceptable hypothesis is found
```

Overall, the simple algorithm sketched above is outperforming the best previous algorithm using optimal parameter values.[7]

4.1.1. *Simulation*

We have verified in simulation that the proposed algorithm works in practice, both in the LF1 and LF2 setting using the rate $R = \frac{1}{3}$ concatenated repetition code.

5. Algorithm Description

Having introduced the key idea in a simplistic manner, we now formalize it by stating a new five-step LPN solving algorithm (see Algorithm 2) in detail. Its first three steps combine several well-known techniques on this problem, i.e., changing the distribution of secret vector [25], sorting and merging to make the dimension of samples shorter [3], and partial secret guessing [4], together. The efficiency improvement comes from

[7]Adopting the same method to implement their overlapping steps, for the $(160, \frac{1}{10})$ LPN instance, the Bernstein–Lange algorithm and the new algorithm cost $2^{39.43}$ and $2^{35.50}$ bit operations, respectively. Thus, the latter offers an improvement with a factor roughly 16 to solve this small-scale instance.

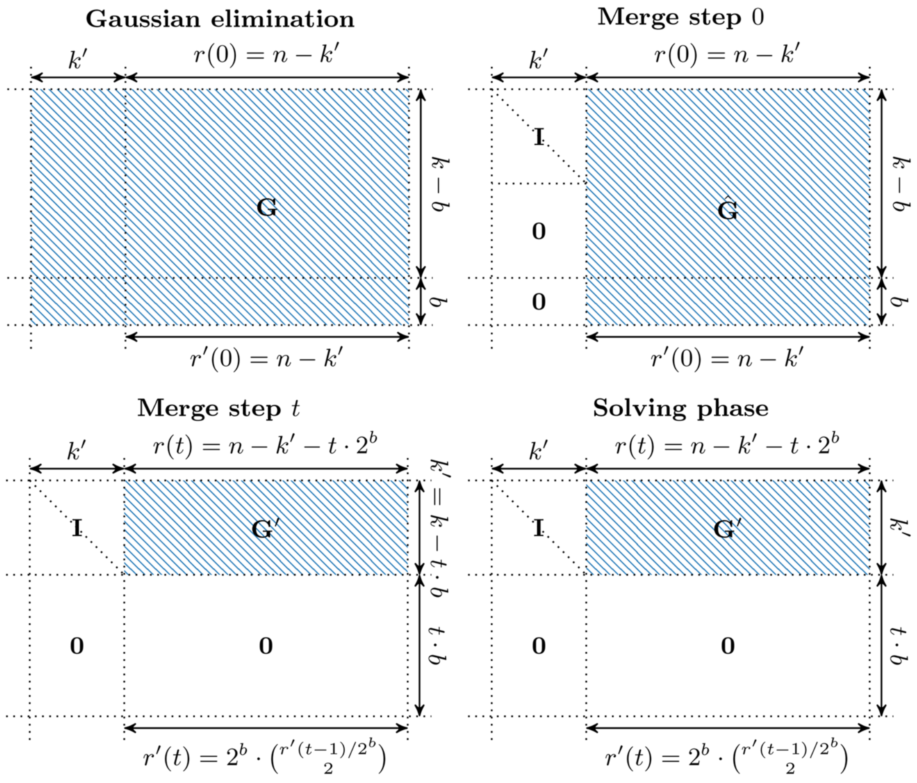

Fig. 2. Here, we illustrate the different steps of the new algorithm, using the LF1 and the LF2 merging approaches. In the figure, we only show the upper systematic part used in hypothesis testing.

a novel idea introduced in the last two subsections—if we employ a linear covering code and rearrange samples according to their nearest codewords, then the columns in the matrix subtracting their corresponding codewords lead to sparse vectors desired in the distinguishing process. We later propose a new distinguishing technique—subspace hypothesis testing, to remove the influence of the codeword part using fast Walsh–Hadamard transform. The algorithm consists of five steps, each described in separate subsections. These steps are graphically illustrated in Figs. 2 and 3.

5.1. *Gaussian Elimination*

Recall that our LPN problem is given by $\mathbf{z} = \mathbf{x}\mathbf{G} + \mathbf{e}$, where $\mathbf{z}$ and $\mathbf{G}$ are known. We can apply an arbitrary column permutation π without changing the problem (but we change the error locations). A transformed problem is $\pi(\mathbf{z}) = \mathbf{x}\pi(\mathbf{G}) + \pi(\mathbf{e})$. This means that we can repeat the algorithm many times using different permutations, which very much resembles the operation of information-set decoding algorithms.

Continuing, we multiply by a suitable $k \times k$ matrix $\mathbf{D}$ to bring the matrix $\mathbf{G}$ to a systematic form, $\hat{\mathbf{G}} = \mathbf{D}\mathbf{G}$. The problem remains the same, except that the unknowns

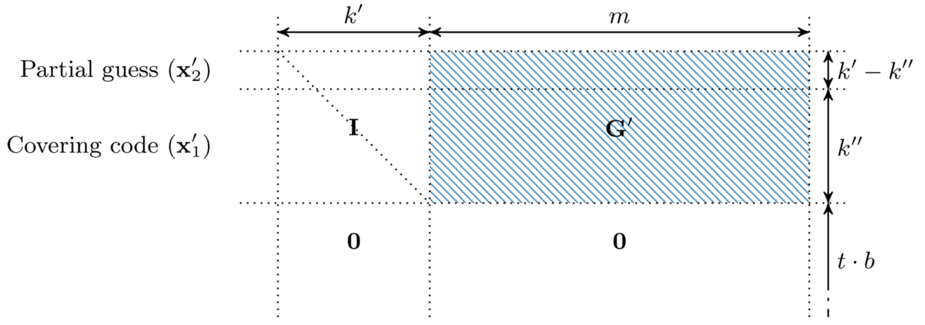

Fig. 3. After the columns have been merged t times, we have a matrix as shown above. In the upper part, we perform the partial secret guessing. The remaining part will be projected (with distortion) into a smaller space of dimension l using a covering code.

are now given by the vector $\tilde{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1}$. This is just a change of variables. As a second step, we also add the codeword $\begin{pmatrix} z_1 & z_2 & \cdots & z_k \end{pmatrix} \hat{\mathbf{G}}$ to our known vector $\mathbf{z}$, resulting in a received vector starting with k zero entries. Altogether, this corresponds to the change $\hat{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1} + \begin{pmatrix} z_1 & z_2 & \cdots & z_k \end{pmatrix}$.

Our initial problem has been transformed, and the problem is now written as

$$\hat{\mathbf{z}} = \begin{pmatrix} \mathbf{0} & \hat{z}_{k+1} & \hat{z}_{k+2} & \cdots & \hat{z}_n \end{pmatrix} = \hat{\mathbf{x}}\hat{\mathbf{G}} + \mathbf{e}, \tag{13}$$

where now $\hat{\mathbf{G}}$ is in systematic form. Note that these transformations do not affect the noise level. We still have a single noise variable added in every position.

5.1.1. *Time–Memory Trade-Off*

Schoolbook implementation of the above Gaussian elimination procedure requires about $\frac{1}{2} \cdot n \cdot k^2$ bit operations; we propose, however, to reduce its complexity by using a more sophisticated time–memory trade-off technique. We store intermediate results in tables and then derive the final result by adding several items in the tables together. The detailed description is as follows.

For a fixed s, divide the matrix $\mathbf{D}$ in $a = \lceil \frac{k}{s} \rceil$ parts, i.e.,

$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_1 & \mathbf{D}_2 & \ldots & \mathbf{D}_a \end{pmatrix}, \tag{14}$$

where $\mathbf{D}_i$ is a sub-matrix with s columns (except possibly the last matrix $\mathbf{D}_a$). Then store all possible values of $\mathbf{D}_i\mathbf{x}^{\mathrm{T}}$ for $\mathbf{x} \in \mathbb{F}_2^s$ in tables indexed by i, where $1 \leq i \leq a$. For a vector $\mathbf{g} = \begin{pmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \ldots & \mathbf{g}_a \end{pmatrix}$, the transformed vector is

$$\mathbf{D}\mathbf{g}^{\mathrm{T}} = \mathbf{D}_1\mathbf{g}_1^{\mathrm{T}} + \mathbf{D}_2\mathbf{g}_2^{\mathrm{T}} + \ldots + \mathbf{D}_a\mathbf{g}_a^{\mathrm{T}}, \tag{15}$$

where $\mathbf{D}_i\mathbf{g}_i^{\mathrm{T}}$ can be read directly from the table.

The cost of constructing the tables is about $\mathcal{O}(2^s)$, which can be negligible if memory in the later merge step is much larger. Furthermore, for each column, the transformation costs no more than $k \cdot a$ bit operations; so, this step requires

$$C_1 = (n-k)\cdot k \cdot a < n \cdot k \cdot a \tag{16}$$

bit operations in total if 2^s is much smaller than n.

5.1.2. *A Minor Improvement*

One observation is that only the distribution of the first $k' = k - t \cdot b$ entries in the secret vector affects the later steps. In other words, we just need to make the first k' entries biased. Thus, we can ignore the Gaussian elimination processing on the bottom tb rows of the $\mathbf{G}$. More formally, let the first k columns of $\mathbf{G}$ be an invertible matrix $\mathbf{G}_0$, where

$$\mathbf{G}_0 = \begin{bmatrix} \mathbf{G}_{01} & \mathbf{G}_{02} \\ \mathbf{G}_{03} & \mathbf{G}_{04} \end{bmatrix},$$

then instead of setting $\mathbf{D} = \mathbf{G}_0^{-1}$, we define

$$\mathbf{D} = \begin{bmatrix} \mathbf{G}_{01}^{-1} & \mathbf{0} \\ -\mathbf{G}_{03}\mathbf{G}_{01}^{-1} & \mathbf{I} \end{bmatrix}.$$

Then, the first k' column of $\mathbf{DG}$ is of the form

$$\begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix}.$$

Denote the transformed secret vector $\hat{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1} + \mathbf{z}_{[1,\ldots,k']}$ similarly. Then, we have that $\hat{\mathbf{z}} = \hat{\mathbf{x}}\hat{\mathbf{G}} + \mathbf{e}$, where $\hat{\mathbf{G}}$ is $\mathbf{DG}$ and,

$$\hat{\mathbf{z}} = \left(\mathbf{0}\ \hat{z}_{k'+1}\ \hat{z}_{k'+2} \cdots \hat{z}_n\right) = \mathbf{z} + \mathbf{z}_{[1,\ldots,k']}\hat{\mathbf{G}} \tag{17}$$

Using the space-time trade-off technique, the complexity can be computed as:

$$C_1' = (n-k')\cdot k \cdot \left\lceil \frac{k'}{s} \right\rceil < n \cdot k \cdot \left\lceil \frac{k'}{s} \right\rceil. \tag{18}$$

Compared with Eq. (16), we reduce the value a from $\lceil \frac{k}{s} \rceil$ to $\lceil \frac{k'}{s} \rceil$, where $k' = k - t \cdot b$.

5.2. *Merging Columns*

This next step consists of merging columns. The input to this step is $\hat{\mathbf{z}}$ and $\hat{\mathbf{G}}$. We write $\hat{\mathbf{G}} = \left(\mathbf{I}\ \mathbf{L}_0\right)$ and process only the matrix $\mathbf{L}_0$. As the length of $\mathbf{L}_0$ is typically much larger than the systematic part of $\hat{\mathbf{G}}$, this is roughly no restriction at all. We then use the a sort-and-match technique as in the BKW algorithm, operating on the matrix $\mathbf{L}_0$. This process will give us a sequence of matrices denoted $\mathbf{L}_0, \mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_t$.

Let us denote the number of columns of $\mathbf{L}_i$ by $r(i)$, with $r(0) = r'(0) = n - k'$. Adopting the LF1 type technique, every step operating on columns will reduce the number of samples by 2^b, yielding that

$$m = r(t) = r(0) - t \cdot 2^b \iff n - k' = m + t \cdot 2^b. \tag{19}$$

Using the setting of LF2, the number of samples is

$$\begin{aligned} m = r'(t) &= 2^b \cdot \binom{r'(t-1)/2^b}{2} \\ \implies n - k' &\approx \sqrt[2^{t+1}]{2^{(b+1)(2^{t+1}-1)} \cdot m}. \end{aligned} \tag{20}$$

The expression for $r'(t)$ does not appear in [29], but it can be found in [5]. We see that if m is equal to $3 \cdot 2^b$, the number of samples preserves during the reductions. Implementations suggest that there is no visible effect on the success of the algorithm,[8] so we adopt this setting.

Apart from the process of creating the $\mathbf{L}_i$ matrices, we need to update the received vector in a similar fashion. A simple way is to put $\hat{\mathbf{z}}$ as a first row in the representation of $\hat{\mathbf{G}}$. This procedure will end with a matrix $\left(\mathbf{I}\ \mathbf{L}_t\right)$, where $\mathbf{L}_t$ will have all $t \cdot b$ last entries in each column all zero. By discarding the last $t \cdot b$ rows, we have a given matrix of dimension $k - t \cdot b$ that can be written as $\mathbf{G}' = \left(\mathbf{I}\ \mathbf{L}_t\right)$, and we have a corresponding received vector $\mathbf{z}' = \left(\mathbf{0}\ z_1'\ z_2' \cdots z_m'\right)$. The first $k' = k - t \cdot b$ positions are only affected by a single noise variable, so we can write

$$\mathbf{z}' = \mathbf{x}'\mathbf{G}' + \left(e_1\ e_2 \cdots e_{k'}\ \tilde{e}_1\ \tilde{e}_2 \cdots \tilde{e}_m\right), \tag{21}$$

for some unknown $\mathbf{x}'$ vector (here, we remove the bottom $t \cdot b$ bits of $\hat{\mathbf{x}}$ to form the length k' vector $\mathbf{x}'$), where

$$\tilde{e}_i = \sum_{i_j \in \mathcal{T}_i,\ |\mathcal{T}_i| \le 2^t} e_{i_j} \tag{22}$$

and $\mathcal{T}_i$ contains the positions that have been added up to form the $(k'+i)$th column of $\mathbf{G}'$. By the piling-up lemma, the bias for $\tilde{e}_i$ increases to ϵ^{2^t}. We denote the complexity of this step by C_2, where

$$C_2 = \begin{cases} \sum_{i=1}^{t} (k+1-i \cdot b) \cdot (n - k' - i \cdot 2^b), & \text{the LF1 setting,} \\ \sum_{i=1}^{t} (k+1-i \cdot b) \cdot (n - k'), & \text{the LF2 setting.} \end{cases}$$

In the both cases

$$C_2 \approx (k+1) \cdot t \cdot n. \tag{23}$$

[8]This is first pointed out in [29].

5.3. *Partial Secret Guessing*

The previous procedure outputs $\mathbf{G}'$ with dimension $k' = k - t \cdot b$ and m columns. We now divide $\mathbf{x}'$ into two parts:

$$\mathbf{x}' = \left(\mathbf{x}_1' \ \mathbf{x}_2'\right), \tag{24}$$

where $\mathbf{x}_1'$ is of length k''. In this step, we simply guess all vectors $\mathbf{x}_2 \in \mathcal{B}_2(k' - k'', w_0)$ for some w_0 and update the observed vector $\mathbf{z}'$ accordingly. This transforms the problem to that of attacking a new smaller LPN problem of dimension k'' with the same number of samples. Firstly, note that this will only work if $w_{\mathrm{H}}\left(\mathbf{x}_2'\right) \leq w_0$, and we denote this probability by $P(w_0, k' - k'')$. Secondly, we need to be able to distinguish a correct guess from incorrect ones and this is the task of the remaining steps. The complexity of this step is

$$C_3 = m \cdot \sum_{i=0}^{w_0} \binom{k' - k''}{i} i. \tag{25}$$

5.4. *Covering-Coding Method*

In this step, we use a $[k'', l]$ linear code $\mathcal{C}$ with covering radius d_C to group the columns. That is, we rewrite

$$\mathbf{g}_i' = \mathbf{c}_i + \mathbf{e}_i', \tag{26}$$

where $\mathbf{c}_i$ is the nearest codeword in $\mathcal{C}$, and $w_{\mathrm{H}}\left(\mathbf{e}_i'\right) \leq d_C$. The employed linear code is characterized by a systematic generator matrix

$$\mathbf{F} = \left(\mathbf{I} \ \mathbf{A}\right) \in \mathbb{F}_2^{l \times k''}, \tag{27}$$

that has the corresponding parity-check matrix

$$\mathbf{H} = \left(\mathbf{A}^{\mathrm{T}} \ \mathbf{I}\right) \in \mathbb{F}_2^{(k''-l) \times k''}. \tag{28}$$

There are several ways to select a code. An efficient way of realizing the above grouping idea is by a table-based syndrome-decoding technique. The procedure is as follows:

1. We construct a constant-time query table containing $2^{k''-l}$ items, in each of which stores the syndrome and its corresponding minimum-weight error vector.
2. If the syndrome $\mathbf{H}\mathbf{g}_i'^{\mathrm{T}}$ is computed, we then find its corresponding error vector $\mathbf{e}_i'$ by checking in the table; adding them together yields the nearest codeword $\mathbf{c}_i$.

The remaining task is to calculate the syndrome efficiently. We sort the vectors $\mathbf{g}_i'$ according to the first l bits, where $0 \leq i \leq m$, and group them into 2^l partitions denoted by $\mathcal{P}_j$ for $1 \leq j \leq 2^l$. Starting from the partition $\mathcal{P}_1$ whose first l bits are all zero, we can derive the syndrome by reading its last $k'' - l$ bits without any additional computational cost. If we know one syndrome in $\mathcal{P}_j$, we then can compute another syndrome in the same partition within $2(k''-l)$ bit operations, and another in a different partition whose first l-bit vector has Hamming distance 1 from that of $\mathcal{P}_j$ within $3(k''-l)$ bit operations. Therefore, the complexity of this step is

$$C_4 = (k''-l) \cdot (2m + 2^l). \tag{29}$$

Notice that the selected linear code determines the syndrome table, which can be pre-computed within complexity $\mathcal{O}\left(k'' \cdot 2^{k''-l}\right)$. For some instances, building such a full syndrome table may dominate the complexity, i.e., when $k'' \cdot 2^{k''-l}$ becomes too large. Here, we use a code concatenation to reduce the size of the syndrome table, thereby making this cost negligible compared with the total attacking complexity.

We split the search space into two (or several) separate spaces by using a concatenated code construction. As an example, let $\mathcal{C}'$ be a concatenation of two $[k''/2, l/2]$ linear codes. Then, the syndrome tables can be built in $\mathcal{O}\left(k'' \cdot 2^{k''/2-l/2}\right)$ time and memory. Assuming that the two codes are identical and they will both contribute to the final noise. The decoding complexity then changes to

$$C_4' = (k''-l) \cdot (2m + 2^{l/2}). \tag{30}$$

5.5. *Subspace Hypothesis Testing*

In the subspace hypothesis testing step, we group the (processed) samples $(\mathbf{g}_i', z_i')$ in sets $L(\mathbf{c}_i)$ according to their nearest codewords and define the function $f_L(\mathbf{c}_i)$ as

$$f_L(\mathbf{c}_i) = \sum_{(\mathbf{g}_i', z_i') \in L(\mathbf{c}_i)} (-1)^{z_i'}. \tag{31}$$

The employed systematic linear code $\mathcal{C}$ describes a bijection between the linear space $\mathbb{F}_2^l$ and the set of all codewords in $\mathbb{F}_2^{k''}$, and moreover, due to its systematic feature, the corresponding information vector appears explicitly in their first l bits. We can thus define a new function

$$g(\mathbf{u}) = f_L(\mathbf{c}_i), \tag{32}$$

such that $\mathbf{u}$ represents the first l bits of $\mathbf{c}_i$ and exhausts all points in $\mathbb{F}_2^l$.

The Walsh transform of g is defined as

$$G(\mathbf{v}) = \sum_{\mathbf{u} \in \mathbb{F}_2^l} g(\mathbf{u})(-1)^{\langle \mathbf{v}, \mathbf{u} \rangle}. \tag{33}$$

Here, we exhaust all candidates of $\mathbf{v} \in \mathbb{F}_2^l$ by computing the Walsh transform.

The following lemma illustrates the reason why we can perform hypothesis testing on the subspace $\mathbb{F}_2^l$.

Lemma 2. *There exits a unique vector* $\mathbf{v} \in \mathbb{F}_2^l$ *s.t.,*

$$\langle \mathbf{v}, \mathbf{u} \rangle = \left\langle \mathbf{x}', \mathbf{c}_i \right\rangle. \tag{34}$$

Proof. As $\mathbf{c}_i = \mathbf{uF}$, we obtain

$$\left\langle \mathbf{x}', \mathbf{c}_i \right\rangle = \mathbf{x}'(\mathbf{uF})^{\mathrm{T}} = \mathbf{x}'\mathbf{F}^{\mathrm{T}}\mathbf{u}^{\mathrm{T}} = \left\langle \mathbf{x}'\mathbf{F}^{\mathrm{T}}, \mathbf{u} \right\rangle. \tag{35}$$

Thus, we construct the vector $\mathbf{v} = \mathbf{x}'\mathbf{F}^{\mathrm{T}}$ that fulfills the requirement. On the other hand, the uniqueness is obvious. □

Before we continue to go deeper into the details of the attack, we will now try to illustrate how the subspace hypothesis test is performed. Consider the following.

$$\left(\begin{array}{c} * \\ \vdots \\ * \\ \hline y'_i \\ \hline * \\ \vdots \end{array}\right)^{\mathrm{T}} = \underbrace{\left(\begin{array}{c} x'_1 \\ \vdots \\ x'_{k''} \\ \hline 0 \\ \vdots \end{array}\right)^{\mathrm{T}}}_{\text{Secret } \mathbf{x}'_1} \underbrace{\left(\begin{array}{c|c|c} * & g'_1 & * \\ \vdots & \vdots & \vdots \\ * & g'_{k''} & * \\ \hline & 0 & \\ & \vdots & \end{array}\right)}_{\text{Query matrix}} \overset{\text{Rewrite } \mathbf{g}'_i \text{ as codeword } \mathbf{c}_i = \mathbf{uF} \text{ and discrepancy } \mathbf{e}'_i}{=} \left(\begin{array}{c} x'_1 \\ \vdots \\ x'_{k''} \\ \hline 0 \\ \vdots \end{array}\right)^{\mathrm{T}} \left(\begin{array}{c|c|c} * & (\mathbf{uF} + \mathbf{e}'_i)_1 & * \\ \vdots & \vdots & \vdots \\ * & (\mathbf{uF} + \mathbf{e}'_i)_{k''} & * \\ \hline & 0 & \\ & \vdots & \end{array}\right).$$

As a next step, we can separate the discrepancy $\mathbf{e}'_i$ from $\mathbf{u}'\mathbf{F}$, which yields

$$\left(\begin{array}{c} x'_1 \\ \vdots \\ x'_{k''} \\ \hline 0 \\ \vdots \end{array}\right)^{\mathrm{T}} \left(\begin{array}{cc|c|c} * & * & (\mathbf{uF})_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & (\mathbf{uF})_{k''} & * \\ \hline & & 0 & \\ & & \vdots & \end{array}\right) = \left(\begin{array}{c} * \\ \vdots \\ * \\ \hline y'_i + \left\langle \mathbf{x}'_1, \mathbf{e}'_i \right\rangle \\ \hline * \\ \vdots \end{array}\right)^{\mathrm{T}}.$$

We now see that the dimension of the problem has been reduced, i.e., $\mathbf{x}'_1\mathbf{F}^{\mathrm{T}} \in \mathbb{F}_2^l$, where $l < k''$. A simple transformation yields

$$\left(\begin{array}{c} (\mathbf{x}'_1\mathbf{F}^{\mathrm{T}})_1 \\ \vdots \\ (\mathbf{x}'_1\mathbf{F}^{\mathrm{T}})_l \\ \hline 0 \\ \vdots \end{array}\right)^{\mathrm{T}} \left(\begin{array}{cc|c|c} * & * & u_1 & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & u_l & * \\ \hline & & 0 & \\ & & \vdots & \end{array}\right) = \left(\begin{array}{c} * \\ \vdots \\ * \\ \hline y'_i + \left\langle \mathbf{x}'_1, \mathbf{e}'_i \right\rangle \\ \hline * \\ \vdots \end{array}\right)^{\mathrm{T}}.$$

Since $w_{\mathrm{H}}\left(\mathbf{e}_i'\right) \le d_C$ and $w_{\mathrm{H}}\left(\mathbf{x}_i'\right) \approx \eta \cdot k''$, the contribution from $\left\langle \mathbf{x}_1', \mathbf{e}_i' \right\rangle$ is small. Note that $\mathbf{e}_i'$ is the error from the above procedure, and that we did not include the error from the oracle and the merging procedure. Recall that the sequence received from oracle is $z_i = y_i + e_i$, that after merging the columns of $\mathbf{G}$ becomes $z_i' = y_i' + \tilde{e}_i$. All things considered (all sources of error piled on the sequence), we have

$$z_i' + \left\langle \mathbf{x}', \mathbf{e}_i \right\rangle = y_i + \tilde{e}_i + \left\langle \mathbf{x}_1', \mathbf{e}_i' \right\rangle . \tag{36}$$

Given the candidate $\mathbf{v}$, $G(\mathbf{v})$ is the difference between the number of predicted 0's and the number of predicted 1's for the bit $\tilde{e}_i + \left\langle \mathbf{x}', \mathbf{e}_i' \right\rangle$. Assume that $\left\langle \mathbf{x}_1', \mathbf{e}_i' \right\rangle$ will contribute to a noise with bias no smaller then ϵ_{set}. If $\mathbf{v}$ is the correct guess, then it is Bernoulli distributed with noise parameter

$$\frac{1}{2} \cdot \left(1 + \epsilon^{2^t} \cdot \epsilon_{set}\right); \tag{37}$$

otherwise, it is considered random. Thus, the best candidate $\mathbf{v}_{\mathrm{opt}}$ is the one that maximizes the absolute value of $G(\mathbf{v})$, i.e.,

$$\mathbf{v}_{\mathrm{opt}} = \underset{\mathbf{v} \in \mathbb{F}_2^l}{\arg\max} \, |G(\mathbf{v})|, \tag{38}$$

and we need approximately

$$\frac{4 \ln 2 \cdot l}{(\epsilon^{2^t} \cdot \epsilon_{set})^2}. \tag{39}$$

samples[9] to distinguish these two cases.

Note that a false positive can be recognized without much cost. If the distinguisher fails, we then choose another permutation to run the algorithm again. The procedure will continue until we find the secret vector $\mathbf{x}$.

We use the fast Walsh–Hadamard transform technique to accelerate the distinguishing step. As the hypothesis testing runs for every guess of $\mathbf{x}_2'$, the overall complexity of this step is

$$C_5 \overset{\text{def}}{=} l \cdot 2^l \cdot \sum_{i=0}^{w_0} \binom{k' - k''}{i}. \tag{40}$$

6. Analysis

In the previous section, we already indicated the complexity of each step. We now put it together in a single complexity estimate. We first formulate the formula for the possibility

[9]This estimation follows results from linear cryptanalysis [8,30]. In the proceeding's version [16], we use a too optimistic estimation on the required number of samples, i.e., using a constant factor before the term $\frac{1}{\epsilon^2}$. This rough estimation also appears in some previous work.

of having at most w errors in j positions $P(w, j)$, which follows a binomial distribution, i.e.,

$$P(w, j) \overset{\text{def}}{=} \sum_{i=0}^{w} \binom{j}{i} (1-\eta)^{j-i} \cdot \eta^{i}. \tag{41}$$

The complexity consists of three parts:

– **Inner complexity** The complexity of each step in the algorithm, i.e.,

$$C_{one-iter} = C_1' + C_2 + C_3 + C_4 + C_5. \tag{42}$$

These steps will be performed every iteration.

– **Guessing** The probability of making a correct guess on the weight of $\mathbf{x}_2'$, i.e.,

$$P_{\text{guess}} \overset{\text{def}}{=} \mathbf{Pr}\left[w_{\text{H}}\left(\mathbf{x}_2'\right) \leq w_0\right] = P(w_0, k' - k''). \tag{43}$$

– **Testing** The probability that the constraint on the bias level introduced by coding (i.e., no smaller than ϵ_{set}) is fulfilled, denoted by P_{test}.

The success probability in one iteration is $P(w_0, k' - k'') \cdot P_{\text{test}}$. The presented algorithm is of the Las Vegas type, and in each iteration, the complexity accumulates step by step. Hence, the following theorem is revealed.

Theorem 1. (The complexity of Algorithm 2) *Let n be the number of samples required and $a, b, t, k'', l, w_0, \epsilon_{set}$ be algorithm parameters. For the* LPN *instance with parameter (k, η), the number of bit operations required for a successful run of the new attack, denoted $C^*(a, b, t, k'', l, w_0, \epsilon_{set})$, is equal to*

$$\begin{aligned} P_{\text{guess}}^{-1} \cdot P_{\text{test}}^{-1} \cdot \Bigg\{ & k \cdot n \cdot a + (k+1) \cdot t \cdot n \\ & + \sum_{i=0}^{w_0} \binom{k'-k''}{i} (m \cdot i + l \cdot 2^l) + (k'' - l) \cdot (2m + 2^l) \Bigg\}, \end{aligned} \tag{44}$$

under the condition that

$$m \geq \frac{4 \ln 2 \cdot l}{(\epsilon^{2^t} \cdot \epsilon_{set})^2}, \tag{45}$$

where $m = n - t2^b$ in the LF1 *setting and $m = n = 3 \cdot 2^b$ in the* LF2 *setting.*[10]

Proof. The complexity of one iteration is given by $C_1' + C_2 + C_3 + C_4 + C_5$. The expected number of iterations is the inverse of $P_{\text{guess}} \cdot P_{\text{test}}$. Substituting the formulas

[10]The accurate value of m should be $n - k' - t2^b$ in the LF1 setting and $n - k'$ in the LF2 setting. We take this approximation as k' is negligible compared with n.

into the above will complete the proof. Condition (45) ensures that we have enough samples to determine the correct guess with high probability. $\square$

The remaining part is to calculate the value of P_{test}, which is determined by the employed code.

6.1. *Bias from a Single Perfect Code*

If we use a length k'' perfect code[11] with covering radius d_C, the bias ϵ' in $\mathbf{e}'_i$ is determined by the following lemma.[12]

Proposition 1. (Bias from covering code [31]) *If the covering code* $\mathbf{F}$ *has an optimal covering radius, then the probability* $\mathbf{Pr}_{w_H(\mathbf{x}'_1)=c}\left[\langle \mathbf{x}'_1, \mathbf{e}'_i \rangle = 1\right]$ *is given by*

$$\varphi(c) \overset{\text{def}}{=} |\mathcal{B}_2(k'', d_C)|^{-1} \cdot \sum_{i \text{ odd}}^{\min(c,d_C)} \binom{c}{i} \cdot |\mathcal{B}_2(k''-c, d_C-i)| \tag{46}$$

where k'' is the dimension of $\mathbf{x}'_1$ and d_C is the covering radius. Thus, the computed bias $\epsilon(c)$ conditioned on the weight of $\mathbf{x}'_1$ is

$$\epsilon(c) = 1 - 2\varphi(c). \tag{47}$$

Proof. Let the c nonzero positions of $\mathbf{x}'_1$ represent a set of bins and the $k''-c$ zero positions another set of bins.

$$\underbrace{\sqcup\ \sqcup\ \cdots\ \sqcup}_{c}\ \Big|\ \underbrace{\sqcup\ \sqcup\ \sqcup\ \sqcup\ \cdots\ \sqcup}_{k''-c}$$

Assume that a bin contains at most one ball. If there is an odd number of balls in the c bins, then $\langle \mathbf{x}'_1, \mathbf{e}'_i \rangle = 1$. Suppose that there are i balls. Then, there are $\binom{c}{i}$ ways to arrange the balls within those bins. In total, we may place up to $j \overset{\text{def}}{=} \min(c, d_C)$ balls, so there remains up to $j - i$ balls to be placed in the other set of $k''-c$ bins, which counts to $|\mathcal{B}_2(k''-c, d_C-i)|$ possibilities. The summation includes all odd i. $\square$

The bias function $\epsilon(c)$ is monotonically decreasing. If we preset a bias level ϵ_{set}, all the possible $\mathbf{x}'_1$ with weight no more than c_0 will be distinguished successfully, where $c_0 = \min\{c | \epsilon(c) \geq \epsilon_{set}\}$. We can then present a lower bound on P_{test}, i.e.,

$$P_{\text{test}} = P(c_0, k'').$$

[11] In the sequel, we assume that when the code length is relatively large, it is reasonable to approximate a perfect code by a random liner code. We replace the covering radius by the sphere-covering bound to estimate the expected distance d, i.e., d is the smallest integer, s.t. $\sum_{i=0}^{d} \binom{k''}{i} \geq 2^{k''-l}$. We give more explanation in Sect. 9.

[12] We would like to thank Sonia Bogos and Serge Vaudenay for pointing out this accurate bias computation.

Note that this estimation lower bounds the success probability, which is higher in practice as the distinguisher will still succeed with certain probability even if the bias level introduced by coding is smaller than the one we set. We can also make use of the list-decoding idea to increase the success probability by keeping a small list of candidates.

6.2. *The Concatenated Construction*

Until now, we have only considered to use a single code for the covering code part. In some cases, performing syndrome decoding may be too expensive for optimal parameters and to overcome this, we need to use a concatenated code construction. As an example, we will illustrate the complexity estimation for the concatenation of two codes, which is the optimal code construction for solving several LPN instances.

As in the previous case, we set an explicit lower bound on the bias $\epsilon' \geq \epsilon_{set}$ introduced from the covering code part, which is attained only by a certain set $\mathcal{E}_{\epsilon_{set}}$ of (good) error patterns in the secret. For a concatenation of two codes, we have divided the vector into two parts

$$\mathbf{x}'_1 = \left(\bar{\mathbf{x}}_1 \ \bar{\mathbf{x}}_2\right) \tag{48}$$

and hence,

$$\mathbf{e}'_i = \left(\bar{\mathbf{e}}_i^{(1)} \ \bar{\mathbf{e}}_i^{(2)}\right). \tag{49}$$

The noise $\langle \mathbf{x}'_1, \mathbf{e}'_i \rangle$ can be rewritten as

$$\langle \mathbf{x}'_1, \mathbf{e}'_i \rangle = \left\langle \bar{\mathbf{x}}_1, \bar{\mathbf{e}}_i^{(1)} \right\rangle + \left\langle \bar{\mathbf{x}}_2, \bar{\mathbf{e}}_i^{(2)} \right\rangle, \tag{50}$$

which implies that the bias $\epsilon' = \epsilon_1 \epsilon_2$, where ϵ_1 (ϵ_2) is the bias introduced by the first (second) code and can be computed by Proposition 1. We then determine all the (good) error patterns $\mathcal{E}_{\epsilon_{set}}$ in the secret such that the bias $\epsilon' \geq \epsilon_{set}$.

We can write the success probability $P_{\text{test}} \overset{\text{def}}{=} \mathbf{Pr}\left[\mathbf{x}'_1 \in \mathcal{E}_{\epsilon_{set}}\right]$ as

$$\sum_{(\bar{\mathbf{x}}_1 \ \bar{\mathbf{x}}_2) \in \mathcal{E}_{\epsilon_{set}}} \eta^{k''/2 - w_{\text{H}}(\bar{\mathbf{x}}_1)} (1-\eta)^{w_{\text{H}}(\bar{\mathbf{x}}_1)} \cdot \eta^{k''/2 - w_{\text{H}}(\bar{\mathbf{x}}_2)} (1-\eta)^{w_{\text{H}}(\bar{\mathbf{x}}_2)}. \tag{51}$$

We could expect that the algorithm works slightly better in practice, as we discussed before in Sect. 6.1.

The complexity C_4 changes to that in the concatenated code case which we denote by C'_4, and the pre-computation of the syndrome tables has a lowered complexity since the codes are smaller and can be treated separately. Since the pre-computation complexity $\mathcal{O}\left(k'' \cdot 2^{k''/2 - l/2}\right)$ must be less[13] or match the total attacking complexity, the lowered

[13]We could make this cost negligible compared with the total complexity.

Table 2. The bias from a [3,1] repetition code.

$w_{\mathcal{C}_i}$	ϵ
0	1
1	$\frac{1}{2}$
2	0
3	$-\frac{1}{2}$

time complexity allows for looser constraints on the algorithm parameters. Apart from these differences, the complexity expression is the same as that for the non-concatenated construction.

It is straight-forward to extend the above analysis to a concatenation of multiple linear codes. As before, we choose to preset a lower bound ϵ_{set} for the bias and derive a formula to estimate the probability of all the good error patterns in the secret. This type of analysis has actually been done in the toy example from Sect. 4.1.

Example 1. In this toy example from Sect. 4.1, we concatenate 25 [3, 1] repetition codes $\mathcal{C}_i$, for $1 \leq i \leq 25$. For each code $\mathcal{C}_i$, we know that the corresponding bias ϵ is related to the Hamming weight $w_{\mathcal{C}_i}$ of the associated subvector of the secret (as shown in Table 2). In Sect. 4.1, we set the bound for the bias ϵ_{set} to be 2^{-6} and then obtain the success probability[14] in (12).

7. Results

We now present numerical results of the new algorithm attacking three key LPN instances, as shown in Table 3. All aiming for achieving 80-bit security, the first one is with parameter (512, $\frac{1}{8}$), widely accepted in various LPN-based cryptosystems (e.g., HB$^+$ [22], HB$^\#$ [13], LPN-C [14]) after the suggestion from Levieil and Fouque [29]; the second one is with increased length (532, $\frac{1}{8}$), adopted as the parameter of the irreducible Ring-LPN instance employed in Lapin [20]; and the last one is a new design parameter[15] we recommend to use in the future. The attacking details on different protocols will be given later. We note that the new algorithm has significance not only on the above applications but also on some LPN-based cryptosystems without explicit parameter settings (e.g., [10,24]).

7.1. *HB$^+$*

Levieil and Fouque [29] proposed an active attack on HB$^+$ by choosing the random vector $\mathbf{a}$ from the reader to be $\mathbf{0}$. To achieve 80-bit security, they suggested to adjust the lengths of secret keys to 80 and 512, respectively, instead of being both 224. Its

[14] When calculating the success probability in (12), we ignore the probability that a nonzero even number of concatenations have $w_{\mathcal{C}_i} = 3$, since these events are so rare.

[15] This instance requires 2^{81} bit memory using the new algorithm and could withstand all existing attacks on the security level of 2^{80} bit operations.

Table 3. The complexity for solving different LPN instances in the LF2 setting.

LPN instance	Parameters								$\log_2 C^*$
	t	a	b	l	k''	w_0	$-\log_2(\epsilon_{set})$	$\log_2 n$	
$(512, \frac{1}{8})$	5	4	62	60	180	2	14.78	63.6	79.64
$(532, \frac{1}{8})$	5	4	64	62	194	1	15.81	65.6	82.01
$(592, \frac{1}{8})$	5	5	70	66	206	2	18.75	71.6	89.38

security is based on the assumption that the LPN instance with parameter $(512, \frac{1}{8})$ can resist attacks in 2^{80} bit operations. But we solve this instance in $2^{79.64}$ bit operations, indicating that the old parameters are insufficient to achieve 80-bit security.

7.2. *LPN-C and HB#*

Using similar structures, Gilbert et al. proposed two different cryptosystems, one for authentication (HB#) and the other for encryption (LPN-C). By setting the random vector from the reader and the message vector to be both $\mathbf{0}$, we obtain an active attack on HB# authentication protocol and a chosen-plaintext-attack on LPN-C, respectively. As their protocols consist of both secure version (RANDOM-HB# and LPN-C) and efficient version (HB# and Toeplitz LPN-C), we need to analyze them separately.

7.2.1. *Using Toeplitz Matrices*

Toeplitz matrix is a matrix in which each ascending diagonal from left to right is a constant. Thus, when employing a Toeplitz matrix as the secret, if we attack its first column successively, then only one bit in its second column is unknown. So the problem is transformed to that of solving a new LPN instance with parameter $(1, \frac{1}{8})$. We then deduce the third column, the fourth column, and so forth. The typical parameter settings of the number of the columns (denoted by m) are 441 for HB#, and 80 (or 160) for Toeplitz LPN-C. In either case, the cost for determining the vectors except for the first column is bounded by 2^{40}, negligible compared with that of attacking one $(512, \frac{1}{8})$ LPN instance. Therefore, for achieving the security of 80 bits, these efficient versions that use Toeplitz matrices should use a larger LPN instance.

7.2.2. *Random Matrix Case*

If the secret matrix is chosen totally at random, then there is no simple connection between different columns to exploit. One strategy is to attack column by column, thereby deriving an algorithm whose complexity is that of attacking a $(512, \frac{1}{8})$ LPN instance multiplied by the number of the columns. That is, if $m = 441$, then the overall complexity is about $2^{88.4}$. We may slightly improve the attack by exploiting that the different columns share the same random vector in each round.

7.3. *Lapin with an Irreducible Polynomial*

Heyse et al. [20] use a $(532, \frac{1}{8})$ Ring-LPN instance with an irreducible polynomial[16] to achieve 80-bit security. We show here that this parameter setting is not secure enough for Lapin to thwart attacks on the level of 2^{80}. Although the new attack on a $(532, \frac{1}{8})$ LPN instance requires approximately 2^{82} bit operations, larger than 2^{80}, there are two key issues to consider:

– Ring-LPN is believed to be no harder than the standard LPN problem. For the instance in Lapin using a quotient ring modulo the irreducible polynomial $x^{532} + x + 1$, it is possible to optimize the procedure by further taking advantage of the ring structure, thereby resulting in a more efficient attack than the generic one.
– The definition of bit complexity here poorly characterizes the actual computational hardness as the computer can parallel many bit operations in one clock cycle. We believe that a better definition should be a vectorized version, i.e., defining the "atomic" operation as the addition or multiplication between two 64 (or 128)-bit vectors. The refined definition is a counterpart of that in the Advanced Encryption Standard (AES), where 80-bit security means that we can perform 2^{80} AES encryptions, not just bit operations. If we adopt this vectorized security definition, the considered Lapin instance is far away from achieving 80-bit security.

We suggest to increase the size of the employed irreducible polynomial in Lapin for 80-bit security.

8. Experiments

We show the experimental results in this part, using a [46, 24] linear code that is a concatenation of two binary [23, 12] Golay codes[17] for the subspace hypothesis testing procedure.

8.1. *Validation of Success Rates*

Starting with $2^{25.6}$ LPN samples, we run two groups of simulations with k equal to 142 and 166, respectively. The noise rate η varies to achieve a reasonable success probability. We perform 4 BKW steps with size 24 for the prior and include one more step for the latter. Moreover, we stick to the LF2 type reduction steps for a better performance.

The comparison between the simulation results and their theoretical counterparts is shown in Table 4. The simulated values are obtained by running about 200 trials for each LPN instance. Meanwhile, as we always keep about $2^{25.6}$ samples after each reduction step, the number of samples for the statistical testing procedure is also approximately

[16]The Lapin instantiation with a reducible polynomial designed for 80-bit security has been broken within about 2^{71} bit operations in [17].

[17]Binary [23, 12] Golay codes are perfect codes with optimal covering property. The concatenation of two Golay codes can produce a larger linear code with fairly good covering property and also efficient decoding. Moreover, the implementation of Golay codes is simple and well studied.

Table 4. Success probability in simulation v.s. in theory.

η ($k = 166$)	0.070	0.075	0.080	0.085	0.090
Simulation results	0.982	0.896	0.783	0.546	0.286
Theoretical estimation	0.959	0.872	0.694	0.443	0.265
η ($k = 142$)	0.115	0.125	0.135	0.145	0.155
Simulation results	0.928	0.780	0.570	0.345	0.117
Theoretical estimation	0.904	0.772	0.538	0.277	0.060

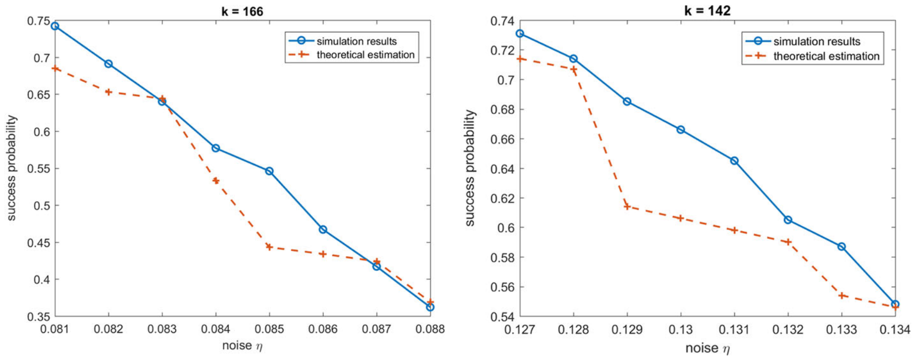

Fig. 4. Fine-grained success probability comparison .

$2^{25.6}$. Thus, we can compute the theoretical success probabilities according to Proposition 1, Equations (39) and (51).

We conclude from Table 4 that the adopted theoretical estimation is a conservative estimation as discussed in Sect. 6.1, since the simulation results are almost always better than the theoretical ones. On the other hand, the theoretical predictions are fairly close to our experimental results. This understanding is further consolidated in Fig. 4 plotting the success probability comparison with fine-grained choices of the noise rate η and more accurate simulated probabilities, i.e., we run 1000 tries for each LPN instance.

8.2. *The Largest Instance*

We solve the $(136, \frac{1}{4})$ LPN instance in 12 h on average using one thread of a server with Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz and 256 GB of RAM. This solved instance is slightly larger than the $(135, \frac{1}{4})$ one reported in [12] requiring 64-thread parallel computating of 13.84 days by using the Well-Pooled MMT algorithm and 5.69 days by using the Hybrid algorithm,[18] on a server with 256 GB of RAM. Though it is tricky to compare implementations of different types of algorithms, our results support that the BKW variants are more efficient when the noise rate η is high.

In the implementation, we ask the LPN oracle for around $2^{31.6}$ samples and then perform three LF2 type BKW steps with size 30. After this step, we have zeroed out

[18]The Well-Pooled MMT algorithm is an ISD variant and the Hybrid algorithm combines solving ideas from ISD and BKW.

90 positions and do the subspace hypothesis testing on the remaining 46 positions via employing a concatenation of two [23, 12] Golay codes. We run 12 trails in approximately 48 h and succeed 4 times.

9. More on the Covering-Coding Method

In this section, we describe more aspects of the covering-coding technique, thus emphasizing the most novel and essential step in the new algorithm.

9.1. *Sphere-Covering Bound*

We use sphere-covering bound to estimate the bias ϵ' contributed by the new technique for two reasons. Firstly, there is a well-known conjecture [7] in coding theory that the covering density approaches 1 asymptotically if the code length goes to infinity. Thus, it is sensible to assume that the linear code has a good covering radius, when the code length k'' is relatively large. Secondly, we could see from the previous example that the desired key feature is a linear code with low average error weights, which is smaller than its covering radius. From this perspective, the covering bound brings us a good estimation.

9.2. *Attacking Public-Key Cryptography*

We know various decodable covering codes that could be employed in the new algorithm, e.g., table-based syndrome decodable linear codes, concatenated codes built on Hamming codes, Golay codes and repetition codes, etc.. For the aimed cryptographic schemes in this paper, i.e., HB variants, LPN-C, and Lapin with an irreducible polynomial, the first three are efficient, but in the realm of public-key cryptography (e.g., schemes proposed by Alekhnovich [2], Damgård and Park [9], Duc and Vaudenay [11]), the situation alters. For these systems, their security is based on LPN instances with huge secret length (tens of thousands) and extremely low error probability (less than half a percent), so due to the competitive average weight of the error vector shown by the previous example in Sect. 4.1, the concatenation of repetition codes with much lower rate seems more applicable—by low-rate codes, we remove more bits when using the covering-coding method.

9.3. *Alternative Collision Procedure*

Although the covering-coding method is employed only once in the new algorithm, we could derive numerous variants, and among them, one may find a more efficient attack. For example, we could replace several steps in the later stage of the collision procedure by adding two vectors decoded to the same codeword together. This alternative technique is similar to that invented by Lamberger et al. [27,28] for finding near-collisions of hash function. By this procedure, we could eliminate more bits in one step at the cost of increasing the error rate; this is a trade-off, and the concrete parameter setting should be analyzed more thoroughly later.

Actually, with the help of this alternative collision idea, a series of recent papers [1,18,19,26] have greatly reduced the solving complexity of the LWE problem, the q-ary counterpart of LPN, both asymptotically and concretely. But we failed to find better attacks when applying this idea to the LPN instances of cryptographic interests in the proposed authentication protocols and LPN-C, since the noise rates are high. We believe that this idea could be useful when the noise is relatively small and leave this problem as an interesting scope for future research.

10. Conclusions

In this paper, we have described a new algorithm for solving the LPN problem that employs an approximation technique using covering codes together with a subspace hypothesis testing technique to determine the value of linear combinations of the secret bits. Complexity estimates show that the algorithm beats all the previous approaches, and in particular, we can present academic attacks on instances of LPN that has been suggested in different cryptographic primitives.

There are a few obvious improvements for this new technique, one being the use of strong distinguishers and another one being the use of more powerful constructions of good codes. There are also various modified versions that need to be further investigated. One such idea as described in Sect. 9.3 is to use the new technique inside a BKW step, thereby removing more bits in each step at the expense of introducing another contribution to the bias. An interesting open problem is whether these ideas can improve the asymptotic behavior of the BKW algorithm.

Acknowledgements

Open access funding provided by Lund University. The authors would like to thank the anonymous ASIACRYPT 2014 reviewers for their helpful comments. They also would like to thank Sonia Bogos and Serge Vaudenay for their suggestions on analyzing the complexity more accurately. This work was supported in part by the Swedish Research Council (Grant No. 621-2012-4259 and No. 2015-04528). Qian Guo was also supported in part by the Erasmus Mundus Action 2 Scholarship, by the National Natural Science Foundations of China (Grant No. 61170208) and Shanghai Key Program of Basic Research (Grant No. 12JC1401400), and by the Norwegian Research Council (Grant No. 247742/070).

References

[1] M.R. Albrecht, J.C. Faugère, R. Fitzpatrick, L. Perret, Lazy Modulus switching for the BKW algorithm on LWE, in H. Krawczyk, editor, *Public-Key Cryptography—PKC 2014*. Lecture Notes in Computer Science, vol. 8383 (Springer Berlin, 2014), pp. 429–445

[2] M. Alekhnovich, More on average case versus approximation complexity, in *FOCS* (IEEE Computer Society, 2003), pp. 298–307

[3] A. Blum, A. Kalai, H. Wasserman, Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, **50**(4), 506–519 (2003)

[4] D. Bernstein, T. Lange, Never trust a bunny, in *Radio Frequency Identification Security and Privacy Issues* (Springer, Berlin, 2013), pp. 137–148

[5] S. Bogos, F. Tramer, S. Vaudenay, On Solving LPN using BKW and Variants. Tech. rep., Cryptology ePrint Archive, Report 2015/049 (2015)

[6] S. Bogos, S. Vaudenay, Optimization of lpn solving algorithms, in *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4–8, 2016, Proceedings, Part I 22* (Springer, 2016) pp. 703–728

[7] G. Cohen, I. Honkala, S. Litsyn, A. Lobstein, *Covering Codes* (Elsevier, Amsterdam, 1997)

[8] T.M. Cover, J.A. Thomas, *Elements of Information Theory* (Wiley, New York, 2012)

[9] I. Damgård, S. Park, Is Public-Key Encryption Based on LPN Practical? Cryptology ePrint Archive, Report 2012/699 (2012). http://eprint.iacr.org/

[10] Y. Dodis, E. Kiltz, K. Pietrzak, D. Wichs, Message authentication, revisited, in D. Pointcheval, T. Johansson, editors, *EUROCRYPT 2012*. LNCS, vol. 7237 (Springer, Heidelberg, 2012), pp. 355–374

[11] A. Duc, S. Vaudenay, HELEN: a public-key cryptosystem based on the LPN and the decisional minimal distance problems, in *AFRICACRYPT 2013* (Springer, Berlin, 2013), pp. 107–126

[12] A. Esser, R. Kübler, A. May, LPN decoded, in J. Katz, H. Shacham, editors, *Advances in Cryptology—CRYPTO 2017—37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II*. Lecture Notes in Computer Science, vol. 10402 (Springer, 2017), pp. 486–514

[13] H. Gilbert, M.J.B. Robshaw, Y. Seurin, $HB^{\#}$: Increasing the security and the efficiency of HB^{+}, in N.P. Smart, editors, *EUROCRYPT 2008*. LNCS, vol. 4965 (Springer, Heidelberg, 2008), pp. 361–378

[14] H. Gilbert, M.J.B. Robshaw, Y. Seurin, How to encrypt with the LPN problem, in L. Aceto, I. Damgård, L.A. Goldberg, M.M. Halldorsson, A. Ingolfsdottir, I. Walukiewicz, editors, *ICALP 2008, Part II*. LNCS, vol. 5126 (Springer, Heidelberg, 2008), pp. 679–690

[15] H. Gilbert, M.J.B. Robshaw, H. Sibert, An Active Attack Against HB^{+}—A Provably Secure Lightweight Authentication Protocol. Cryptology ePrint Archive, Report 2005/237 (2005). http://eprint.iacr.org/

[16] Q. Guo, T. Johansson, C. Löndahl, Solving LPN using covering codes, in *Advances in Cryptology—ASIACRYPT 2014* (Springer, 2014), pp. 1–20

[17] Q. Guo, T. Johansson, C. Löndahl, A new algorithm for solving ring-LPN with a reducible polynomial. *IEEE Trans. Inf. Theory*, **61**(11), 6204–6212 (2015)

[18] Q. Guo, T. Johansson, P. Stankovski, Coded-BKW: solving LWE using lattice codes, in *Advances in Cryptology—CRYPTO 2015* (Springer, 2015), pp. 23–42

[19] Q. Guo, T. Johansson, E.Mårtensson, P. Stankovski, Coded-bkw with sieving, in T. Takagi, T. Peyrin, editors, *Advances in Cryptology—ASIACRYPT 2017—23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3–7, 2017, Proceedings, Part I*. Lecture Notes in Computer Science, vol. 10624 (Springer, 2017), pp. 323–346

[20] S. Heyse, E. Kiltz, V. Lyubashevsky, C. Paar, K. Pietrzak, Lapin: an efficient authentication protocol based on ring-LPN, in *FSE 2012* (2012), pp. 346–365

[21] N.J. Hopper, M. Blum, Secure human identification protocols, in C. Boyd, editor, *ASIACRYPT 2001*. LNCS, vol. 2248 (Springer, Heidelberg, 2001), pp. 52–66

[22] A. Juels, S.A. Weis, Authenticating pervasive devices with human protocols, in V. Shoup, editor, *CRYPTO 2005*. LNCS, vol. 3621 (Springer, Heidelberg, 2005), pp. 293–308

[23] J. Katz, J.S. Shin, Parallel and concurrent security of the HB and HB^{+} protocols, in S. Vaudenay, editor, *EUROCRYPT 2006*. LNCS, vol. 4004 (Springer, Heidelberg, 2006), pp. 73–87

[24] E. Kiltz, K. Pietrzak, D. Cash, A. Jain, D. Venturi, Efficient authentication from hard learning problems, in K.G Paterson, editor, *EUROCRYPT 2011*. LNCS, vol. 6632 (Springer, Heidelberg, 2011) pp. 7–26
[25] P. Kirchner, Improved Generalized Birthday Attack. Cryptology ePrint Archive, Report 2011/377 (2011). http://eprint.iacr.org/
[26] P. Kirchner, P.A. Fouque, An improved BKW algorithm for LWE with applications to cryptography and lattices, in *Advances in Cryptology—CRYPTO 2015* (Springer, 2015), pp. 43–62
[27] M. Lamberger, F. Mendel, V. Rijmen, K. Simoens, Memoryless near-collisions via coding theory. *Des. Codes Cryptogr.* **62**(1), 1-18 (2012)
[28] M. Lamberger, E. Teufl, Memoryless near-collisions, revisited. *Inf. Process. Lett.*, **113**(3), 60-66 (2013)
[29] E. Levieil, P.A. Fouque, An improved LPN algorithm, in *Proceedings of SCN 2006*. LNCS 4116 (Springer, Heidelberg, 2006), pp. 348–359
[30] A.A. Selçuk, On probability of success in linear and differential cryptanalysis. *J. Cryptol.* **21**(1), 131–147 (2008)
[31] S. Vaudenay, *Private Communication*
[32] B. Zhang, L. Jiao, M. Wang, Faster algorithms for solving LPN, in *EUROCRYPT 2016* (Springer, 2016), pp. 168–195