Solving LPN Using Covering Codes

Qian Guo[1⋆], Thomas Johansson[2], and Carl Löndahl[2⋆⋆]

[1] Dept. of Electrical and Information Technology, Lund University, Lund, Sweden
& School of Computer Science, Fudan University, Shanghai, China
qian.guo@eit.lth.se
[2] Dept. of Electrical and Information Technology, Lund University, Lund, Sweden
{thomas.johansson,carl.londahl}@eit.lth.se

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 instances suggested for 80-bit security in cryptographic schemes like HB variants, LPN-C and Lapin, in less than 2^{80} operations.

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[3], the LPN problem is a 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 makes it ideal for lightweight cryptography.

The first time the LPN problem was employed in a cryptographic construction was in the Hopper-Blum (HB) identification protocol [17]. 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 [18], and Katz and Shin [19] 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

⋆ Supported in part by the National Natural Science Foundations of China (Grants No. 61170208) and Shanghai Key Program of Basic Research (Grant No. 12JC1401400).
⋆⋆ Supported by the Swedish Research Council (Grants No. 621-2012-4259).
[3] LPN with adversarial error is $\mathcal{NP}$-hard.

et al. [14] that the HB$^+$ protocol is vulnerable to active attacks, i.e. *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. [12] 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.

In [13], Gilbert et al. proposed a way to use LPN in encryption of messages, which resulted in the cryptosystem LPN-C. Kiltz et al. [22] and Dodis et al. [9] 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. [16], and an LPN-based encryption scheme called Helen, proposed by Duc and Vaudenay [10]. 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[4] 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 are removed, the remaining are exhausted 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 [26] proposed to use Fast Walsh-Hadamard Transform in the BKW algorithm when searching for the secret. In an unpublished paper, Kirchner [23] suggested to transform the problem into systematic form, where each information (key) bit then appears as an observed symbol, pertubated by noise. This requires the adversary to only exhaust the biased noise variables rather than the key bits. When the error rate is low, the noise variable search space is very small and this technique decreases the attack complexity. Building on the work by Kirchner [23], Bernstein and Lange [5] 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 [26], which were suggested as design parameters of LPN-C [13] for 80-bit security.

In this paper, we propose a new algorithm for solving the LPN problem based on [23, 5]. 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

[4] For a fixed error rate.

Table 1. Comparison of different algorithms for solving LPN with parameters $(512, 1/8)$.

Algorithm	Complexity ($\log_2$)		
	Queries	Time	Memory
Levieil-Fouque [26]	75.7	87.5	84.8
Bernstein-Lange [5]	68.6	85.7	77.6
New algorithm	66.3	79.9	75.3

comparison to »state-of-the-art« algorithms and we manage to break previously unbroken parameters of HB variants, Lapin and LPN-C. As an example, we attack the common $(512, 1/8)$-instance of LPN and break its 80-bit security barrier. A comparision of complexity of different algorithms[5] is shown in Table 1.

The organization of the paper is as follows. In Section 2, we give some preliminaries and introduce the LPN problem in detail. Moreover, in Section 3 we give a short description of the BKW algorithm. We briefly describe the general idea of our new attack in Section 4 and more formally in Section 5. In Section 6, we analyze its complexity. The results when the algorithm is applied on various LPN-based cryptosystems are given in Section 7 and in Section 8, we describe some aspects of the covering-coding technique. Section 9 concludes the paper.

2 The LPN Problem

We will 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 $\mathbf{Pr}[X = 0] = 1/2\,(1 + \epsilon)$. Let k be a security parameter and let $\mathbf{x}$ be a binary vector of length k.

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

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

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 $\mathbf{x}$.

[5] The Bernstein-Lange algorithm is originally proposed for Ring-LPN, and by a slight modification [5], one can apply it to the LPN instances as well. It shares the beginning steps (i.e., the steps of Gaussian elimination and the collision procedure) with the new algorithm, so for a fair comparison, we use the same implementation of these steps when computing their complexity.

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*

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

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 is written as $\mathbf{z} = \mathbf{x}\mathbf{G} + \mathbf{e}$, where $\mathbf{z} = \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix}$, and the matrix $\mathbf{G}$ is formed as $\mathbf{G} = \begin{bmatrix} \mathbf{g}_1^{\mathrm{T}} & \mathbf{g}_2^{\mathrm{T}} & \cdots & \mathbf{g}_n^{\mathrm{T}} \end{bmatrix}$. 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* $X_1, X_2, ...X_n$ *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,*

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

3 The BKW Algorithm

The BKW algorithm is due to Blum, Kalai and Wasserman [3]. In the spirit of generalized birthday algorithms, their approach uses an iterative sort-and-match procedure on the columns of the generator matrix $\mathbf{G}$, which iteratively reduces the dimension of $\mathbf{G}$.

Initially, one searches for all combinations of two columns in $\mathbf{G}$ that add to zero in the last b entries. Assume that one finds two columns $\mathbf{g}_{i_1}^{\mathrm{T}}, \mathbf{g}_{i_2}^{\mathrm{T}}$ such that

$$\mathbf{g}_{i_1} + \mathbf{g}_{i_2} = [* * \cdots * \underbrace{0\, 0 \cdots 0}_{b \text{ symbols}}],$$

where $*$ means any value. Then a new vector $\mathbf{g}_1^{(2)} = \mathbf{g}_{i_1} + \mathbf{g}_{i_2}$ is formed. An "observed symbol" is also formed, corresponding to this new column by forming $z_1^{(2)} = z_{i_1} + z_{i_2}$. If $y_1^{(2)} = \left\langle \mathbf{x}, \mathbf{g}_1^{(2)} \right\rangle$, then $z_1^{(2)} = y_1^{(2)} + e_1^{(2)}$, where now $e_1^{(2)} = e_{i_1} + e_{i_2}$. It can be verified that $\mathbf{Pr}\left[e_1^{(2)} = 0\right] = 1/2(1 + \epsilon^2)$.

There are two approaches to realize the above merging procedure. One, raised by Blum et al. [3], called *LF1* type by Levieil and Fouque [26], and later adopted by Bernstein and Lange [5], is choosing one sample in each partition with the same last b entries, and then adding it to the remaining samples in the same partition. Thus, the number of samples reduces by about 2^b after this operation. The other method is a heuristic called *LF2* in [26], which computes any pair with the same last b entries. It produces more samples at the cost of increased dependency, thereby gaining more efficiency in practice but losing rigorous analysis in theory. We will use the *LF1* setting throughout the remaining part of the paper.

Put all such new columns in a matrix $\mathbf{G}_2$,

$$\mathbf{G}_2 = \left[\mathbf{g}_1^{(2)\mathrm{T}}\ \mathbf{g}_2^{(2)\mathrm{T}}\ \cdots\ \mathbf{g}_{n-2^b}^{(2)\mathrm{T}}\right].$$

If n is the number of columns in $\mathbf{G}$, then the number of columns in $\mathbf{G}_2$ will be $n - 2^b$. Note that the last b entries of every column in $\mathbf{G}_2$ are all zero. In connection to this matrix, the vector of observed symbols is

$$\mathbf{z}_2 = \left[z_1^{(2)}\ z_2^{(2)}\ \cdots\ z_{n-2^b}^{(2)}\right],$$

where $\mathbf{Pr}\left[z_i^{(2)} = y_i^{(2)}\right] = 1/2(1+\epsilon^2)$, for $1 \leq i \leq n - 2^b$.

We now iterate the same, picking one column and then adding it to another suited 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 $t-2$ times will reduce the number of unknown variables to $k - bt$ in the remaining problem.

For each iteration the noise level is squared. By the piling-up lemma we have that

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

Hence, the bias decreases quickly to low levels. The remaining unknown key variables are guessed and for each guess we check whether the bias is present or not. The procedure is summarized in Algorithm 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, and a graphical interpretation of the key step is given in Appendix A.

Assume that we have an initial LPN problem described by $\mathbf{G} = \left[\mathbf{g}_1^{\mathrm{T}}\ \mathbf{g}_2^{\mathrm{T}}\ \cdots\ \mathbf{g}_n^{\mathrm{T}}\right]$ and $\mathbf{z} = \mathbf{x}\mathbf{G} + \mathbf{e}$, where $\mathbf{z} = \left[z_1\ z_2\ \cdots\ z_n\right]$, where $z_i = y_i + e_i = \langle \mathbf{x}, \mathbf{g}_i \rangle + e_i$.

As previously shown in [23] and [5], we may through Gaussian elimination transform $\mathbf{G}$ into systematic form. Assume that the first k columns are linearly

Algorithm 1 BKW Algorithm

Input: Matrix $\mathbf{G}$ with k rows and n columns and received vector $\mathbf{z}$, algorithm parameters b, t

Put the received word as a first row in the matrix, $\mathbf{G}_1 \leftarrow \begin{bmatrix} \mathbf{z} \\ \mathbf{G} \end{bmatrix}$;
for $i = 1$ **to** t **do**
    For $\mathbf{G}_i$, partition the columns by the last $b \cdot i$ bits;
    Form pairs of columns from each partition and form $\mathbf{G}_{i+1}$;
for $\mathbf{x} \in \{0,1\}^{k-bt}$ **do**
    Find the vector $\left[1\ \mathbf{x}\ \mathbf{0}\right]$ such that $\left[1\ \mathbf{x}\ \mathbf{0}\right] \mathbf{G}_{t+1}$ has minimal weight;

independent and forms the matrix $\mathbf{D}$. With a change of variables $\hat{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1}$ we get an equivalent problem description with $\hat{\mathbf{G}} = \left[\mathbf{I}\ \hat{\mathbf{g}}_{k+1}^{\mathrm{T}}\ \hat{\mathbf{g}}_{k+2}^{\mathrm{T}} \cdots\ \hat{\mathbf{g}}_{n}^{\mathrm{T}}\right]$. 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 $\mathbf{G}' = \left[\mathbf{I}\ \mathbf{g}_1'^{\mathrm{T}}\ \mathbf{g}_2'^{\mathrm{T}} \cdots\ \mathbf{g}_m'^{\mathrm{T}}\right]$ and $\mathbf{z}' = \left[\mathbf{0}, z_1', z_2', \ldots z_m'\right]$, where now $\mathbf{G}'$ has dimension $k' \times m$ with $k' = k - bt$ and m is the number of columns remaining after the BKW step. We have $\mathbf{z}' = \mathbf{x}'\mathbf{G}' + \mathbf{e}'$, $\mathbf{Pr}\left[x_i' = 0\right] = 1/2(1+\epsilon)$ and $\mathbf{Pr}\left[\mathbf{x}' \cdot \mathbf{g}_i'^{\mathrm{T}} = z_i\right] = 1/2(1+\epsilon^{2^t})$.

Now we will explain the basics of the new idea proposed in the paper. In a problem instance as above, we may look at the random variables $y_i' = \mathbf{x}' \cdot \mathbf{g}_i'^{\mathrm{T}}$. 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, assume that every column $\mathbf{g}_i'$ would be biased, i.e., every bit in a column position is zero with probability $1/2(1+\epsilon')$. Then we 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, 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 the 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

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

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

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

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

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

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 $m = 1/(\epsilon^{2^t}\epsilon'^c)^2$ samples $z'_1, z'_2, \ldots, z'_m$ and for each guess of the 2^l possible values of $\mathbf{x}''$, we compute how many times $\mathbf{x}'' \cdot \mathbf{u}_i^{\mathrm{T}} = z'_i$ 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 $\mathbf{x}''$, it is an easy task to recover remaining unknown bits of $\mathbf{x}'$.

4.1 An Example Using Dimension $k = 160$

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. 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}.

In the second step we perform $t = 4$ steps of BKW (using the *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 $\mathbf{z}' = \mathbf{x}'\mathbf{G}'$, $\mathbf{Pr}\left[x_i' = 0\right] = 1/2(1+\epsilon)$, where $\epsilon = 0.8$ and $\mathbf{Pr}\left[\mathbf{x}' \cdot \mathbf{g_i'}^{\mathrm{T}} = z_i\right] = 1/2(1+\epsilon^{16})$. So the resulting problem has dimension 75 and the bias is $\epsilon^{2^t} = (0.8)^{16}$.

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[6], 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 3/4 we will change one position and with probability 1/4 we will not have to change any position. In total we expect to change $(3/4 \cdot 1 + 1/4 \cdot 0) \cdot 25$ positions. The expected weight of the length 75 vector $\mathbf{e}_i'$ is 75/4, so the expected bias is $\epsilon' = 1/2$. 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

$$\mathbf{Pr}\left[\mathbf{x}'' \cdot \mathbf{u}_i^{\mathrm{T}} = z_i'\right] = \frac{1}{2}\left(1 + 0.8^{16}\left(\frac{1}{2}\right)^6\right) = \frac{1}{2}(1 + 2^{-11.15}).$$

In the last step we then run through 2^{25} values of $\mathbf{x}''$ and for each of them we compute how often $\mathbf{x}'' \cdot \mathbf{u}_i^{\mathrm{T}} = z_i'$ 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. The probability of having no more than 6 ones in $\mathbf{x}'$ is about 0.37, so we need to repeat the whole process a few times.

In comparison with other algorithms, the best approach we can find is the Kirchner, Bernstein, Lange approach [23, 5], where one can do up to 5 BKW 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.

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

Simulation We have verified in simulation that the proposed algorithm works in practice. We use a rate $R = 1/3$ concatenated repetition code and query the oracle for 2^{24} samples. Simple pruning of the samples with too large distance from the codeword was used to approximate the behaviour of an optimal distinguisher.

[6] 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.

[7] Adopting the same method to implement their overlapping steps, for the $(160, 1/10)$ LPN instance, the Bernstein-Lange algorithm and the new algorithm cost $2^{35.70}$ and $2^{33.83}$ bit operations, respectively. Thus, the latter offers an improvement with a factor roughly 4 to solve this small-scale instance.

Algorithm 2 New attacking algorithm

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, c

```
repeat
    Pick random column permutation π;
    Perform Gaussian elimination on π(G) resulting in G_0 = [I|L_0];
    for i = 1 to t do
        Partition the columns of L_{i-1} by the last b · i bits;
        Denote the set of columns in partition s by L_s;
        Pick a vector a_{is} ∈ L_s;
        for (a ∈ L_s) and (a ≠ a_{is}) do
            L_i ← [L_i|(a + a_{is})];
    Pick a [k'', l] linear code with good covering property;
    Partition the columns of L_t by the middle non-all-zero k'' bits and
    group them by their nearest codewords;
    Set k_1 = k − ab − k'';
    for x'_2 ∈ {0,1}^{k_1} with wt(x'_2) ≤ w_0 do
        Update the observed samples;
        for y ∈ {0,1}^l do
            Use Fast Walsh-Hadamard Transform to compute the
            numbers of 1s and 0s observed respectively;
            Perform hypothesis testing whose threshold is defined as a
            function of c;
until acceptable hypothesis is found
```

The average execution time is ~ 1.86 seconds on an Apple iMac 3.06 GHz Intel Core 2 Duo with 4 GB ram running OS X 10.9 (13A603).

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 [23], sorting and merging to make the length of samples shorter [3], and partial secret guessing [5], together. The efficiency improvement comes from 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.

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.

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 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 $\left[z_1\ z_2 \cdots\ z_k\right]\hat{\mathbf{G}}$ to our known vector $\mathbf{z}$, resulting in a received vector starting with k zero entries. Altogether, this corresponds to the change $\hat{\mathbf{x}} = \mathbf{x}\mathbf{D}^{-1} + \left[z_1\ z_2 \cdots\ z_k\right]$.

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

$$\hat{\mathbf{z}} = \left[\mathbf{0}\ \hat{z}_{k+1}\ \hat{z}_{k+2} \cdots\ \hat{z}_n\right] = \hat{\mathbf{x}}\hat{\mathbf{G}} + \mathbf{e}, \tag{1}$$

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.

Schoolbook implementation of the above Gaussian elimination procedure requires about $nk^2/2$ bit-operations; we propose however to reduce its complexity by using a more sophisticated space-time 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 $a = \lceil k/s \rceil$ parts, i.e., $\mathbf{D} = \left[\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_a\right]$, 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 $\mathbf{g} = \left[\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_a\right]$, the transformed vector is

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

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

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

$$C_1 = (n-k) \cdot ka < nka$$

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

5.2 Collision Procedure

This next step contains the BKW part. 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 $r(0) = n - k$. Adopting the *LF1* type technique, every step operating on columns will reduce the number of samples by 2^b, yielding that $m = r(t) = r(0) - t2^b$. 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 tb last entries in each column all zero. By discarding the last tb rows we have a given matrix of dimension $k - tb$ 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 $k' = k - tb$ positions are only affected by a single noise variable, so we can write

$$[\mathbf{0}, \mathbf{z}'] = \mathbf{x}'\hat{\mathbf{G}} + \left[e_1\ e_2 \cdots\ e_{k'}\ \tilde{e}_1\ \tilde{e}_2 \cdots\ \tilde{e}_m\right], \tag{2}$$

for some unknown $\mathbf{x}'$ vector, where $\tilde{e}_i = \sum_{i_j \in \mathcal{T}_i, |\mathcal{T}_i| \leq 2^t} e_{i_j}$ 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 C_2, where

$$C_2 = \sum_{i=1}^{t} (k + 1 - ib)(n - i2^b) \approx (k + 1)tn.$$

5.3 Partial Secret Guessing Procedure

The previous procedure outputs $\mathbf{G}'$ with dimension $k' = k - tb$ and $m = n - k - t2^b$ columns. We removed the bottom tb bits of $\hat{\mathbf{x}}$ to form the length k' vector $\mathbf{x}'$, with $\mathbf{z}' = \mathbf{x}'\mathbf{G}' + \tilde{\mathbf{e}}$.

We now divide $\mathbf{x}'$ into two parts: $\mathbf{x}' = \left[\mathbf{x}'_1\ \mathbf{x}'_2\right]$, where $\mathbf{x}'_1$ is of length k''. In this step, we simply guess all vectors $\mathbf{x}_2 \in \mathbb{F}_2^{k'-k''}$ such that $wt(\mathbf{x}_2) \leq w_0$ 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 $wt(\mathbf{x}_2) \leq w_0$, 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

$$C_3 = m \sum_{i=0}^{w_1} \binom{k' - k''}{i} i.$$

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

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

where $\mathbf{c}_i$ is the nearest codeword in $\mathcal{C}$, and $wt(\mathbf{e}'_i) \leq d_C$. The employed linear code is characterized by a systematic generator matrix $\mathbf{F} = \left[\mathbf{I}\ \mathbf{F}'\right]_{l \times k''}$; we thus obtain a corresponding parity-check matrix $\mathbf{H} = \left[\mathbf{F}'^{\mathrm{T}}\ \mathbf{I}\right]_{(k''-l)\times k''}$.

There are several ways to select a code. One 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 query time 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, according to the first l bits, sort the vectors $\mathbf{g}'_i$, 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 $\mathcal{P}_j$, then we 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

$$C_4 = (k'' - l)(2m + 2^l).$$

Notice that the selected linear code determines the syndrome table, which can be pre-computed within complexity $\mathcal{O}(k''2^{k''-l})$. The optimal parameter suggests that this cost is acceptable compared with the total attacking complexity.

The expected distance to the nearest codeword determines the bias ϵ' in $\mathbf{e}'_i$. This plays important roles in the later hypothesis testing step: if we rearrange the columns $\mathbf{e}'_i$ as a matrix, then it is sparse; therefore, we can view the ith value in one column as a random variable R_i distributed according to $\mathsf{Ber}_{\frac{d}{k''}}$, where d is the expected distance. We can bound it by the covering radius[8]. Moreover, if the bias is large enough, then it is reasonable to consider R_i, for $1 \leq i \leq i_1$, as independent variables.

5.5 Subspace Hypothesis Testing

Group the 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

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

[8] In the sequel, 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. $\sum_{i=0}^{d} \binom{k''}{i} > 2^{k''-l}$. We give more explanation in Section 8.

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

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

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

The Walsh transform of g is defined as

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

Here we exhaust all candidates of $\mathbf{y} \in \mathbb{F}_2^l$ 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* $\mathbf{y} \in \mathbb{F}_2^l$ *s.t.,*

$$\langle \mathbf{y}, \mathbf{u}\rangle = \langle \mathbf{x}', \mathbf{c}_i\rangle .$$

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

$$\langle \mathbf{x}', \mathbf{c}_i\rangle = \mathbf{x}'\mathbf{F}^{\mathrm{T}}\mathbf{u}^{\mathrm{T}} = \langle \mathbf{x}'\mathbf{F}^{\mathrm{T}}, \mathbf{u}\rangle .$$

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

Given the candidate $\mathbf{y}$, $G(\mathbf{y})$ is the difference between the number of predicted 0 and the number of predicted 1 for the bit $z_i' + \langle \mathbf{x}', \mathbf{c}_i\rangle$. If $\mathbf{y}$ is the correct guess, then it is distributed according to $\mathsf{Ber}_{\frac{1}{2}(1-\epsilon^{2^t}\cdot(\epsilon')^w)}$, where $\epsilon' = 1-\frac{2d}{k''}$ and w is the weight of $\mathbf{x}'$; otherwise, it is considered random. Thus, the best candidate y_0 is the one that maximizes the absolute value of $G(\mathbf{y})$, i.e. $\mathbf{y}_0 = \arg\max_{\mathbf{y}\in F_2^l} |G(\mathbf{y})|$, and we need approximately $1/(\epsilon^{2^{t+1}} \cdot (\epsilon')^{2w})$ samples to distinguish these two cases. Note that false positives are quickly detected in an additional step and this does not significantly increase complexity.

Since the weight w is unknown, we assume that $w \leq c$ and then query for samples. If the assumption is valid, we can distinguish the two distributions correctly; otherwise, we obtain a false positive which can be recognized without much cost, and 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

$$C_5 = l2^l \sum_{i=0}^{w_0} \binom{k'-k''}{i}.$$

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 of having at most w errors in m positions $P(w, m)$ as follows,

$$P(w,m) = \sum_{i=0}^{w} (1-\eta)^{m-i} \eta^i \binom{m}{i}.$$

Therefore, the success probability in one iteration is $P(w_0, k' - k'')P(c, k'')$. In each iteration, the complexity accumulates step by step, hence revealing the following theorem.

Theorem 1 (The complexity of Algorithm 2) *Let n be the number of samples required and a, t, b, w_0, c, l, k'' 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 is equal to $2^{f(k,n,a,t,b,w_0,c,l,k'',\eta)}$, where $f(k, n, a, t, b, w_0, c, l, k'', \eta)$ is a function*[9] *defined as follows,*

$$\begin{aligned}
&f(k,n,a,t,b,w_0,c,l,k'',\eta) = \\
&\log_2 \left(ank + b2^b \frac{t(t+1)(2t+1)}{6} - ((k+1)2^b + nb)\binom{t}{2} + (k+1)tn \right. \\
&\left. + (k''-l)(2(n-t2^b)+2^l) + l2^l \sum_{i=0}^{w_0} \binom{k_1}{i} + (n-t2^b) \sum_{i=0}^{w_0} \binom{k_1}{i} i \right) \\
&- \log_2 \left(\sum_{i=0}^{w_0} (1-\eta)^{k_1-i} \eta^i \binom{k_1}{i} \right) - \log_2 \left(\sum_{i=0}^{c} (1-\eta)^{k''-i} \eta^i \binom{k''}{i} \right) \qquad (3)
\end{aligned}$$

under the condition that

$$n - t2^b > 1/(\epsilon^{2^{t+1}} \cdot (\epsilon')^{2c}), \qquad (4)$$

where $\epsilon = 1 - 2\eta$, $\epsilon' = 1 - \frac{2d}{k''}$ and d is the smallest integer, s.t., $\sum_{i=0}^{d} \binom{k''}{i} > 2^{k''-l}$.

Proof. The complexity in one iteration is $C_1 + C_2 + C_3 + C_4 + C_5$, and the expected number of iterations is the inverse of $P(w_0, k_1)P(c, k'')$; the overall complexity, therefore, is C^*, where

$$C^* = \frac{C_1 + C_2 + C_3 + C_4 + C_5}{P(w_0, k_1)P(c, k'')}.$$

Substituting the detailed formulas into the above expression will end the proof. The condition (4) ensures that we have enough samples to determine the right guess with high probability. □

[9] The symbol k_1 denotes $k - tb - k''$ for notational simplicity.

7 Results

We now present numerical results of the new algorithm attacking three key LPN instances, as shown in Table 2. All aiming for achieving 80-bit security, the first one is with parameter $(512, 1/8)$, widely accepted in various LPN-based cryptosystems (e.g., HB$^+$ [18], HB$^\#$ [12], LPN-C [13]) after the suggestion from Levieil and Fouque [26]; the second one is with increased length $(532, 1/8)$, adopted as the parameter of the irreducible Ring-LPN instance employed in Lapin [16]; and the last one is a new design parameter[10] 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., [9, 22]).

Table 2. The complexity for solving different LPN instances.

LPN instance	Parameters								$\log_2 C^*$
	t	a	b	l	k''	w_0	c	$\log_2 n$	
$(512, 1/8)$	6	9	63	64	124	2	16	66.3	79.92
$(532, 1/8)$	6	9	65	66	130	2	17	68.0	81.82
$(592, 1/8)$	6	10	70	64	137	3	18	72.7	88.07

7.1 HB$^+$

In [26], Levieil and Fouque 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 security is based on the assumption that the LPN instance with parameter $(512, 1/8)$ can resist attacks in 2^{80} bit operations. But we break it in $2^{79.9}$ bit operations, thereby yielding an active attack on 80-bit security of HB$^+$ authentication protocol straightforwardly.

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 separately.

[10] This instance requires $2^{82.3}$ bits memory using the new algorithm, and could withstand all existing attacks on the security level of 2^{80} bit operations.

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 $(1, 1/8)$. 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 $(512, 1/8)$ LPN instance. Therefore, we break the 80-bit security of these »efficient« versions that use Toeplitz matrices.

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 $(512, 1/8)$ LPN instance multiplied by the number of the columns. That is, if $m = 441$, then the overall complexity is about $2^{79.9} \times 441 \approx 2^{88.7}$. 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

In [16], Heyse et al. use a $(532, 1/8)$ Ring-LPN instance with an irreducible polynomial 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 $(532, 1/8)$ LPN instance requires $2^{81.8}$ bit operations, larger than 2^{80}, there are two key issues to consider: 1) the Ring-LPN problem is believed to be not harder than the standard LPN problem[11]; 2) we perform BKW steps using *LF1* setting in the new algorithm, but may obtain a more efficient attack in practice when adopting the *LF2* heuristic, whose effectiveness has been stated and proven in the implementation part of [26]. We suggest to increase the size of the employed irreducible polynomial in Lapin for 80-bit security.

8 More on the Covering-Coding Method

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

Sphere-Covering Bound We use sphere-covering bound, for two reasons, to estimate the bias ϵ' contributed by the new technique. Firstly, there is a well-known conjecture [7] in coding theory, i.e., the covering density approaches 1

[11] For the instance in Lapin using a quotient ring modulo the irreducible polynomial $x^{532} + x + 1$, it is possible to optimize the procedure for inverting a ring element, thereby resulting in a more efficient attack than the generic one.

asymptotically if the code length goes to infinity. Thus, it is sensible to assume for a »good« code, when the code length k'' is relatively large. Secondly, we could see from the previous example that the key feature desired 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.

By concatenating five $[23, 12]$ Golay codes, we construct a $[115, 60]$ linear code[12] with covering radius 15. Its expected weight of error vector is quite close to the sphere-covering bound for this parameter (with gap only 1). We believe in the existence of linear codes with length around 125, rate approximately 1/2 and average error weight that reaches the sphere-covering bound. For explicit code construction, see [15] for details.

Using Soft Information The weight of the error vector $\mathbf{e}'_i$ is different for different values of i, causing the confidence level to vary on different samples. However, the inherent assumption when using Fast Walsh-Hadamard Transform is a constant confidence level over all samples; thus, Fast Walsh-Hadamard Transform is not an optimal distinguishing method. For optimal distinguishing, soft information methods such as likelihood ratio tests are required. We show how to fully exploit soft distinguishing in the longer version of the paper[15].

Attacking Public-Key Cryptography We know various decodable covering codes that could be employed in the new algorithm, e.g., rate about 1/2 linear codes that are table-based syndrome decodable, 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 [1], Damgård and Park [8], Duc and Vaudenay [10]), 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 Section 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.

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 one or two 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. in [24, 25] for finding near-collisions of hash function. By this procedure, we could eliminate more bits in one step

[12] Using this code, we stand at the margin of breaking the 80-bit security of $(512, 1/8)$ LPN instances, with time complexity only $2^{80.5}$ and query complexity $2^{66.2}$.

at the cost of increasing the error rate; this is a trade-off, and the concrete parameter setting should be analyzed more thoroughly later.

9 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.

The new technique has only been described in a rather simplistic manner, due to space limitations. There are a few obvious improvements, one being the use of soft decoding techniques 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 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.

References

1. Alekhnovich, M.: More on Average Case vs Approximation Complexity. In: FOCS. pp. 298–307. IEEE Computer Society (2003)
2. Blum, A., Furst, M., Kearns, M., Lipton, R.: Cryptographic Primitives Based on Hard Learning Problems. In: Stinson, R. (ed.) CRYPTO 1993, LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (1994)
3. Blum, A., Kalai, A., Wasserman, H.: Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model. In: Journal of the ACM, vol. 50, no. 4, pp. 506–519. (2003)
4. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.A.: On the Inherent Intractability of Certain Coding Problems. IEEE Trans. Info. Theory, vol. 24, pp. 384–386. (1978)
5. Bernstein, D., Lange T.: Never trust a bunny. In: Radio Frequency Identification Security and Privacy Issues, pp. 137–148. Springer, Berlin Heidelberg (2013)
6. Chose, P., Joux, A., Mitton, M.: Fast Correlation Attacks: An Algorithmic Point of View. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 209–221. Springer, Heidelberg (2002)
7. Cohen, G., Honkala, I., Litsyn, S., Lobstein, A.: Covering codes. Elsevier, 1997.
8. Damgård, I., Park, S.: Is Public-Key Encryption Based on LPN Practical? Cryptology ePrint Archive, Report 2012/699 (2012), http://eprint.iacr.org/
9. Dodis, Y., Kiltz, E., Pietrzak, K., Wichs, D.: Message Authentication, Revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 355–374. Springer, Heidelberg (2012)
10. Duc, A., Vaudenay, S.: HELEN: A Public-Key Cryptosystem Based on the LPN and the Decisional Minimal Distance Problems. In: AFRICACRYPT 2013. pp. 107–126. Springer, Berlin Heidelberg (2013)

11. Fossorier, M.P.C., Mihaljevic, M.J., Imai, H., Cui, Y., Matsuura, K.: A Novel Algorithm for Solving the LPN Problem and its Application to Security Evaluation of the HB Protocol for RFID Authentication. Cryptology ePrint archive, Report 2012/197 (2012), http://eprint.iacr.org/
12. Gilbert, H., Robshaw, M.J.B., Seurin, Y.: HB$^{\#}$: Increasing the Security and the Efficiency of HB^{+}. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 361–378. Springer, Heidelberg (2008)
13. Gilbert, H., Robshaw, M.J.B., Seurin, Y.: How to encrypt with the LPN problem. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldorsson, M.M., Ingolfsdottir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 679–690. Springer, Heidelberg (2008)
14. Gilbert, H., Robshaw, M.J.B., Sibert, H.: An active attack against HB^{+}—a provably secure lightweight authentication protocol. Cryptology ePrint Archive, Report 2005/237 (2005), http://eprint.iacr.org/
15. Guo, Q., Johansson, T., Löndahl, C.: Solving LPN Using Covering Codes and Soft Information. In preparation.
16. Heyse, S., Kiltz, E., Lyubashevsky, V., Paar, C., Pietrzak, K.: Lapin: An Efficient Authentication Protocol Based on Ring-LPN. In: FSE 2012, pp. 346–365. (2012)
17. Hopper, N.J., Blum, M.: Secure human identification protocols. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 52–66. Springer, Heidelberg (2001)
18. Juels, A., Weis, S.A.: Authenticating pervasive devices with human protocols. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 293–308. Springer, Heidelberg (2005)
19. Katz, J., Shin, J.S.: Parallel and concurrent security of the HB and HB^{+} protocols. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 73–87. Springer, Heidelberg (2006)
20. Katz, J., Shin, J.S., Smith, A.: Parallel and concurrent security of the HB and HB^{+} protocols. Journal of Cryptology 23(3), 402–421 (2010)
21. Kearns., M: Effcient Noise-Tolerant Learning from Statistical Queries. In: J. ACM 45(6), pp. 983–1006. (1998)
22. Kiltz, E., Pietrzak, K., Cash, D., Jain, A., Venturi, D.: Efficient Authentication from Hard Learning Problems. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 7–26. Springer, Heidelberg (2011)
23. Kirchner, P.: Improved Generalized Birthday Attack. Cryptology ePrint Archive, Report 2011/377 (2011), http://eprint.iacr.org/
24. Lamberger, M., Mendel, F., Rijmen, V., Simoens, K.: Memoryless near-collisions via coding theory. Designs, Codes and Cryptography, 62(1): 1-18 (2012)
25. Lamberger, M., Teufl, E.: Memoryless near-collisions, revisited. Information Processing Letters, 113(3): 60-66 (2013)
26. Levieil, E., Fouque, P. A.: An Improved LPN Algorithm. In: Proceedings of SCN 2006, LNCS 4116, pp. 348–359. Springer, Heidelberg (2006)
27. Lyubashevsky, V.: The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem, In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX-RANDOM 2005, LNCS, vol. 3624, pp. 378–389. Springer, Heidelberg (2005)
28. Mitzenmacher, M., Upfal, E.: Probability and computing - randomized algorithms and probabilistic analysis. Cambridge University Press 2005.
29. Munilla, J., Peinado, A.: HB-MP: A further step in the HB-family of lightweight authentication protocols. Computer Networks 51(9), pp. 2262–2267. (2007)

30. Regev, O.: On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing, Proceedings, pp. 84–93. (2005)
31. Stern, J.: A Method for Finding Codewords of Small Weight. In: Wolfmann, J., Cohen, G. (eds.) Coding Theory 1988. LNCS, vol. 388, pp. 106–113. Springer, Heidelberg (1989)
32. Stern, J.: A New Identification Scheme Based on Syndrome Decoding. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 13–21. Springer, Heidelberg (1994)
33. Wagner, D.: A Generalized Birthday Problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–304. Springer, Heidelberg (2002)

A Illustrating the Procedure

In this section, we give an intuitive illustration of subspace hypothesis test performed as follows,

$$\begin{bmatrix} * \\ \vdots \\ * \\ \hline z_i' \\ \hline * \\ \vdots \end{bmatrix} = \underbrace{\begin{bmatrix} x_0 \\ \vdots \\ \hline x_{k''} \\ \hline 0 \\ \vdots \end{bmatrix}^{\mathrm{T}}}_{\text{Secret } \mathbf{x}} \underbrace{\left[\begin{array}{cc|c|c} * & * & g_0 & * \\ \vdots & \vdots & \vdots & \vdots \\ \hline * & * & g_{k''} & * \\ \hline & & 0 & \\ & & \vdots & \end{array}\right]}_{\text{Query matrix}} = \begin{bmatrix} x_0 \\ \vdots \\ \hline x_{k''} \\ \hline 0 \\ \vdots \end{bmatrix}^{\mathrm{T}} \left[\begin{array}{cc|c|c} * & * & (\mathbf{u}'\mathbf{F} + \mathbf{e}_i')_0 & * \\ \vdots & \vdots & \vdots & \vdots \\ \hline * & * & (\mathbf{u}'\mathbf{F} + \mathbf{e}_i')_{k''} & * \\ \hline & & 0 & \\ & & \vdots & \end{array}\right].$$

Rewrite $\mathbf{g}_i$ as codeword $\mathbf{c}_i = \mathbf{u}'\mathbf{F}$ and discrepancy $\mathbf{e}_i'$

We can separate the discrepancy $\mathbf{e}_i'$ from $\mathbf{uF}$, which yields

$$\begin{bmatrix} x_0 \\ \vdots \\ \hline x_{k''} \\ \hline 0 \\ \vdots \end{bmatrix}^{\mathrm{T}} \left[\begin{array}{cc|c|c} * & * & (\mathbf{u}'\mathbf{F})_0 & * \\ \vdots & \vdots & \vdots & \vdots \\ \hline * & * & (\mathbf{u}'\mathbf{F})_{k''} & * \\ \hline & & 0 & \\ & & \vdots & \end{array}\right] = \begin{bmatrix} * \\ \vdots \\ * \\ \hline z_i' + \langle \mathbf{x}, \mathbf{e}_i' \rangle \\ \hline * \\ \vdots \end{bmatrix}.$$

Finally, we note that $\mathbf{x}_1'\mathbf{F}^{\mathrm{T}} \in \mathbb{F}_2^l$, where $l < k''$. A simple transformation yields

$$\begin{bmatrix} (\mathbf{x}_1'\mathbf{F}^{\mathrm{T}})_0 \\ \vdots \\ \hline (\mathbf{x}_1'\mathbf{F}^{\mathrm{T}})_l \\ \hline 0 \\ \vdots \end{bmatrix}^{\mathrm{T}} \left[\begin{array}{cc|c|c} * & * & u_0' & * \\ \vdots & \vdots & \vdots & \vdots \\ \hline * & * & u_l' & * \\ \hline & & 0 & \\ & & \vdots & \end{array}\right] = \begin{bmatrix} * \\ \vdots \\ * \\ \hline z_i' + \langle \mathbf{x}_1', \mathbf{e}_i' \rangle \\ \hline * \\ \vdots \end{bmatrix}.$$

Since $w_{\mathrm{H}}(\mathbf{e}_i') \leq w$, the contribution from $\langle \mathbf{x}_1', \mathbf{e}_i' \rangle$ is very small.