On Solving LPN using BKW and Variants

Implementation and Analysis

Sonia Bogos, Florian Tramèr, and Serge Vaudenay

EPFL
CH-1015 Lausanne, Switzerland
http://lasec.epfl.ch

Abstract. The Learning Parity with Noise problem (LPN) is appealing in cryptography as it is considered to remain hard in the post-quantum world. It is also a good candidate for lightweight devices due to its simplicity. In this paper we provide a comprehensive analysis of the existing LPN solving algorithms, both for the general case and for the sparse secret scenario. In practice, the LPN-based cryptographic constructions use as a reference the security parameters proposed by Levieil and Fouque. But, for these parameters, there remains a gap between the theoretical analysis and the practical complexities of the algorithms we consider. The new theoretical analysis in this paper provides tighter bounds on the complexity of LPN solving algorithms and narrows this gap between theory and practice. We show that for a sparse secret there is another algorithm that outperforms BKW and its variants. Following from our results, we further propose practical parameters for different security levels.

1 Introduction

The Learning Parity with Noise problem (LPN) is a well-known problem studied in cryptography, coding theory and machine learning. In the LPN problem, one has access to queries of the form (v,b), where v is a random vector and the inner product between v and a secret vector s is added to some noise to obtain b. Given these queries, one has to recover the value of s. So, the problem asks to recover a secret vector s given access to noisy inner products of itself with random vectors.

It is believed that LPN is resistant to quantum computers so it is a good alternative to the number-theoretic problems (e.g. factorization and discrete logarithm) which can be solved easily with quantum algorithms. Also, due to its simplicity, it is a nice candidate for lightweight devices. As applications where LPN or LPN variants are deployed, we first have the HB family of authentication protocols: HB [26], HB^+ [27], HB^{++} [11], $\text{HB}^\#$ [20] and AUTH [30]. An LPN-based authentication scheme secure against Man-in-the-Middle was presented in Crypto '13 [34]. There are also several encryption schemes based on LPN: Alekhnovich [3] presents two public-key schemes that encrypt one bit at a time. Later, Gilbert, Robshaw and Seurin [20] introduce LPN-C, a public-key encryption scheme proved to be IND-CPA. Two schemes that improve upon Alekhnovich's scheme are introduced in [15] and [14]. In PKC 2014, Kiltz et al. [29] propose an alternative scheme to [15]. Duc and Vaudenay [17] introduce HELEN, an LPN-based public-key scheme for which they propose concrete parameters for different security levels. A PRNG based on LPN is presented in [8] and [4].

The LPN problem can also be seen as a particular case of the LWE [37] problem where we work in $\mathbb{Z}_2$. While in the case of LWE the reduction from hard lattice problems attests the hardness [37,10,36], in the case of LPN there are no such results. The problem is believed to be hard and is closely related to the long-standing open problem of efficiently decoding random linear codes.

In the current literature, there are few references when it comes to the analysis of LPN. The most well-known algorithm is BKW [9]. When introducing the HB^+ protocol [27], which relies on the hardness of LPN, the authors propose parameters for different levels of security according to the BKW performance. These parameters are shown later to be weaker than thought [32,19]. Fossorier et al. [19] provide a new variant that brings an improvement over the BKW algorithm. Levieil and Fouque [32] also present the BKW algorithm and introduce two improvements over it. For their algorithm based on the fast Walsh-Hadamard transform, they provide the level of security achieved by different instances of LPN. This analysis is referenced by most of the papers that make use of the LPN problem. While they offer a theoretical analysis and propose secure parameters for different levels of security, the authors do not discuss how their theoretical bounds compare to practical results. As we will see, there is a gap between theory and practice. In the domain of machine learning, [21,39] also cryptanalyse the LPN problem. The best algorithm for solving LPN was presented at Asiacrypt 2014 [23]. This new variant of BKW uses covering codes as a novelty.

While these algorithms solve the general case when we have a random secret, in the literature there is no analysis and implementation done for an algorithm specially conceived for the sparse secret case, i.e. the secret has a small Hamming weight.

The BKW algorithm can also be adapted to solve the LWE problem in exponential time. Implementation results and improvements of it were presented in [2,1,16]. In terms of variants of LPN, we have Ring-LPN [24] and Subspace LPN [30]. As an application for Ring-LPN we have the Lapin authentication protocol [24] and its cryptanalysis in [6,22].

Motivation & Contribution. Our paper comes to address exactly the aforementioned open problems. First, we present the current existing LPN solving algorithms in a unified framework. For these algorithms, we provide experimental results and give a better theoretical analysis that brings an improvement over the work of Levieil and Fouque [32]. Furthermore, we implement and analyse three new algorithms for the case where the secret is sparse. Our results show that for a sparse secret the BKW family of algorithms is outperformed by an algorithm that uses Gaussian elimination. Our motivation is to provide a theoretical analysis that matches the experimental results. Although this does not prove that LPN is hard, it gives tighter bounds for the parameters used by the aforementioned cryptographic schemes. It can also be used to have a tighter complexity analysis of algorithms related to LPN solving. Our results were actually used in [23] and also for LWE solving in [16].

Organization. In Section 2 we introduce the definition of LPN and present the main LPN solving algorithms. We also present the main ideas of how the analysis was conducted in [32]. We introduce novel theoretical analyses and show what improvements we bring in Section 3. Besides analysing the current existing algorithms, we propose three new algorithms and analyse their performance in Section 4. In Section 5, we provide the experimental results for the algorithms described in Section 3 & 4. We compare the theory with the practical results and show the tightness of our query complexity. We provide a comparison between all these algorithms in Section 6 and propose practical parameters for a 80 bit security level.

Notations and Preliminaries. Let $\langle\cdot,\cdot\rangle$ denote the inner product, $\mathbb{Z}_2 = \{0,1\}$ and $\oplus$ denote the bitwise XOR. For a domain $\mathcal{D}$, we denote by $x \overset{U}{\leftarrow} \mathcal{D}$ the fact that x is drawn uniformly at random from $\mathcal{D}$. We use small letters for vectors and capital letters for matrices. We denote the Hamming weight of a vector v by $HW(v)$.

2 LPN

In this section we introduce the LPN problem and the algorithms that solve it. For ease of understanding, we present the LPN solving algorithms in a unified framework.

2.1 The LPN Problem

Intuitively, the LPN problem asks to recover a secret vector s given access to noisy inner products of itself and random vectors. More formally, we present below the definition of the LPN problem. We maintain as much as possible the notations from [32].

Definition 1 **(LPN oracle).** *Let $s \xleftarrow{U} \mathbb{Z}_2^k$, let $\tau \in]0, \frac{1}{2}[$ be a constant noise parameter and let Ber_τ be the Bernoulli distribution with parameter τ. Denote by $A_{s,\tau}$ the distribution defined as*

$$\{(v,b) \mid v \xleftarrow{U} \mathbb{Z}_2^k, b = \langle v,s \rangle \oplus d, d \leftarrow Ber_\tau\} \in \mathbb{Z}_2^{k+1}.$$

An LPN oracle $\mathcal{A}_{s,\tau}^{\mathsf{LPN}}$ is an oracle which outputs independent random samples according to $A_{s,\tau}$.

Definition 2 **(Search LPN problem).** *Given access to an LPN oracle $\mathcal{A}_{s,\tau}^{\mathsf{LPN}}$, find the vector s. We denote by $\mathsf{LPN}_{k,\tau}$ the LPN instance where the secret has size k and the noise parameter is τ. Let $k' \leq k$. We say that an algorithm $\mathcal{M}$ (n,t,m,θ,k')-solves the search $\mathsf{LPN}_{k,\tau}$ problem if*

$$\Pr[\mathcal{M}^{\mathcal{A}_{s,\tau}^{\mathsf{LPN}}}(1^k) = s_1 \ldots s_{k'} \mid s \xleftarrow{U} \mathbb{Z}_2^k] \geq \theta,$$

and $\mathcal{M}$ runs in time t, uses memory m and asks at most n queries from the LPN oracle.

Note that we consider here the problem of recovering the first k' bits of the secret. We will show in Section 3 that for all the algorithms we consider, the cost of recovering the full secret s is dominated by the cost of recovering the first k' bits of s.

An equivalent way to formulate the search $\mathsf{LPN}_{k,\tau}$ problem is as follows: given access to a random matrix $A \in \mathbb{Z}_2^{n \times k}$ and a column vector b over $\mathbb{Z}_2$, such that $As \oplus d = b$, find the vector s. Here the matrix A corresponds to the matrix that has the vectors v on its rows, s is the secret vector of size k and b corresponds to the column vector that contains the noisy inner products. The column vector d is of size n and contains the corresponding noise bits.

One may observe that with $\tau = 0$, the problem is solved in polynomial time through Gaussian elimination given $n = \Theta(k)$ queries. The problem becomes hard once noise is added to the inner product. The value of τ can be either independent or dependent of the value k. Usually the value of τ is constant and independent from the value of k. A case where τ is taken as a function of k occurs in the construction of the encryption schemes [3,14]. Intuitively, a larger value of τ means more noise and makes the problem of search LPN harder. The value of the noise parameter is a trade-off between the hardness of the $\mathsf{LPN}_{k,\tau}$ and the practical impact on the applications that rely on this problem.

The LPN problem has also a decisional form. The *decisional* $\mathsf{LPN}_{k,\tau}$ asks to distinguish between the uniform distribution over $\mathbb{Z}_2^{k+1}$ and the distribution $\mathcal{A}_{s,\tau}$. A similar definition for an algorithm that solves decisional LPN can be adopted as above. Let $\mathcal{U}_{k+1}$ denote an oracle that outputs random vectors of size $k+1$. We say that an algorithm $\mathcal{M}$ (n,t,m,θ)-*solves* the decisional $\mathsf{LPN}_{k,\tau}$ problem if

$$\mid \Pr[\mathcal{M}^{\mathcal{A}_{s,\tau}^{\mathsf{LPN}}}(1^k) = 1] - \Pr[\mathcal{M}^{\mathcal{U}_{k+1}}(1^k) = 1] \mid \geq \theta$$

and $\mathcal{M}$ runs in time t, uses memory m and needs at most n queries.

Search and decisional LPN are polynomially equivalent. The following lemma expresses this result.

Lemma 1 **([28,8]).** *If there is an algorithm $\mathcal{M}$ that (n,t,m,θ)-solves the decisional $\mathsf{LPN}_{k,\tau}$, then one can build an algorithm $\mathcal{M}'$ that (n',t',m',θ',k)-solves the search $\mathsf{LPN}_{k,\tau}$ problem, where $n' = O(n \cdot \theta^{-2} \log k)$, $t' = O(t \cdot k \cdot \theta^{-2} \log k)$, $m' = O(m \cdot \theta^{-2} \log k))$ and $\theta' = \frac{\theta}{4}$.*

We do not go into details as this is outside the scope of this paper. We only analyse the solving algorithms for search LPN. From now on we will refer to it simply as LPN.

2.2 LPN Solving Algorithms

In the current literature there are several algorithms to solve the LPN problem. The first that appeared, and the most well known, is BKW [9]. This algorithm recovers the secret s of an $\mathsf{LPN}_{k,\tau}$ instance in sub-exponential $2^{O(\frac{k}{\log k})}$ time complexity by requiring a sub-exponential number $2^{O(\frac{k}{\log k})}$ of queries from the $\mathcal{A}^{\mathsf{LPN}}_{s,\tau}$ oracle. Levieil and Fouque [32] propose two new improvements which are called LF1 and LF2. Fossorier et. al [19] also introduce a new algorithm, which we denote FMICM, that brings an improvement over BKW. The best algorithm to solve LPN was recently presented at Asiacrypt 2014 [23]. It can be seen as a variant of LF1 where covering codes are introduced as a new method to improve the overall algorithm. All these algorithms still require a sub-exponential number of queries and have a sub-exponential time complexity.

Using BKW as a black-box, Lyubashevsky [33] introduces a "pre-processing" phase and solves an $\mathsf{LPN}_{k,\tau}$ instance with $k^{1+\eta}$ queries and with a time complexity of $2^{O(\frac{k}{\log\log k})}$. The queries given to BKW have a worse bias of $\tau' = \frac{1}{2} - \frac{1}{2}\left(\frac{1-2\tau}{4}\right)^{\frac{2k}{\eta \log k}}$. Thus, this variant requires a polynomial number of queries but has a worse time complexity. Given only $n = \Theta(k)$ queries, the best algorithms run in exponential time $2^{\Theta(k)}$ [35,38].

An easy to solve instance of LPN was introduced by Arora and Ge [5]. They show that in the k-wise version where the k-tuples of the noise bits can be expressed as the solution of a polynomial (e.g. there are no 5 consecutive errors in the sequence of queries), the problem can be solved in polynomial time. What makes the problem easy is the fact that an adversary is able to structure the noise.

In this paper we are interested in the BKW algorithm and its improvements presented by Levieil and Fouque [32] and by Guo et al. [23]. The common structure of all these algorithms is the following: given n queries from the $\mathcal{A}^{\mathsf{LPN}}_{s,\tau}$ oracle, the algorithm tries to reduce the problem of finding a secret s of k bits to one where the secret s' has only k' bits, with $k' < k$. This is done by applying several *reduction* techniques. We call this phase the *reduction phase*. Afterwards, during the *solving phase* we can apply a *solving* algorithm that recovers the secret s'. We then update the queries with the recovered bits and restart to fully recover s. For the ease of understanding, we describe all the aforementioned LPN solving algorithms in this setting where we separate the algorithms in two phases. We emphasize the main differences between the algorithms and discuss which improvements they bring.

First, we assume that $k = a \cdot b$. Thus, we can visualise the k-bit length vectors v as a blocks of b bits. We define $\delta = 1 - 2\tau$.

BKW* **Algorithm** The BKW* algorithm as described in [32] works in two phases:

Reduction phase. Given n queries from the LPN oracle, we group them in equivalence classes. Two queries are in the same equivalence class if they have the same value on a set q_1 of b bit positions. These b positions are chosen arbitrarily. There are at most 2^b such equivalence classes. Once this separation is done, we perform the following steps for each equivalence class: pick one query at random, the representative vector, and xor it to the rest of the queries from the same equivalence class. Discard the representative vector. This will give vectors with all bits set to 0 on those b positions. These steps are also illustrated in Algorithm 1 (steps 5 - 10). We are left with at least $n - 2^b$ queries where the secret is reduced to $k - b$ effective bits (others being multiplied by 0 in all queries).

We can repeat the reduction technique $a-1$ times on other disjoint position sets $q_2, \ldots, q_{a-1}$ and end up with at least $n - (a-1)2^b$ queries where the secret is reduced to $k - (a-1)b = b$ bits. The bias of the new queries is $\delta^{2^{a-1}}$, as shown by the following Lemma with $w = 2^{a-1}$.

Lemma 2 ([32,9]). *If $(v_1, b_1), \ldots, (v_w, b_w)$ are the results of w queries from $\mathcal{A}^{\mathsf{LPN}}_{s,p}$, then the probability that:*

$$\langle v_1 \oplus v_2 \oplus \ldots \oplus v_w, s\rangle = b_1 \oplus \ldots \oplus b_w$$

is equal to $\frac{1+\delta^w}{2}$.

It is easy so see that the complexity of performing this step is $O(kan)$.

Algorithm 1 BKW* Algorithm by [32]

1: **Input**: a set V of n queries $(v_i, b_i) \in \{0,1\}^{k+1}$ from the LPN oracle, values a, b such that $k = ab$
2: **Output**: values $s_1, \dots, s_b$

3: Partition the positions $\{1,\dots,k\} \setminus \{1,\dots,b\}$ into disjoint $q_1 \cup \dots \cup q_{a-1}$ with q_i of size b
4: **for** $i = 1$ to $a-1$ **do** ▷ Reduction phase
5:     Partition $V = V_1 \cup \dots \cup V_{2^b}$ s.t. vectors in V_j have the the same bit values on q_i
6:     **foreach** V_j
7:         Choose a random $(v^*, b^*) \in V_j$ as a representative vector
8:         Replace each (v,b) by $(v,b) \oplus (v^*, b^*)$, $(v,b) \in V_j$ for $(v,b) \neq (v^*, b^*)$
9:         Discard (v^*, b^*) from V_j
10:     $V = V_1 \cup \dots \cup V_{2^b}$
11: Discard from V all queries (v,b) such that $HW(v) \neq 1$
12: Partition $V = V_1 \cup \dots \cup V_b$ s.t. vectors in V_j have a bit 1 on position j
13: **foreach** position i ▷ Solving phase
14:     $s_i = \mathsf{majority}(b)$, for all $(v,b) \in V_i$
15: **return** $s_1, \dots, s_b$

After $a-1$ iterations, we are left with at least $n-(a-1)2^b$ queries, and a secret of size of b effective bits at positions $1,\dots,b$. The goal is to keep only those queries that have Hamming weight one (step 11 of Algorithm 1). Given $n-(a-1)2^b$ queries, only $n' = \frac{n-(a-1)2^b}{2^b}$ will have s single non-zero bit on a given position and 0 for the rest of $b-1$ positions. These queries represent the input to the solving phase. The bias does not change since we do not alter the original queries. The complexity for performing this step for $n-(a-1)2^b$ queries is $O(b(n-(a-1)2^b))$ as the algorithm just checks if the queries have Hamming weight 1.

Remark 1. Given that we have performed the xor between pairs of queries, we note that the noise bits are no longer independent. In the analysis of BKW*, this was overlooked by Levieil and Fouque [32].[1] The original BKW [9] algorithm overcomes this problem in the following manner: each query that has Hamming weight 1 is obtained with a fresh set of queries. Given $a2^b$ queries the algorithm runs the xoring process and is left with 2^b vectors. From these 2^b queries, with a probability of $1-\frac{1}{e}$, there is one with Hamming weight 1 on a given position i. In order to obtain more such queries the algorithm repeats this process with fresh queries. This means that for guessing 1 bit of the secret, the original algorithm requires $n = a \cdot 2^b \cdot \frac{1}{1-1/e} \cdot n'$ queries, where n' denotes the number of queries needed for the solving phase. This is larger than $n = 2^b n' + (a-1)2^b$ which is the number of queries given by Levieil and Fouque [32]. We implemented and run BKW* as described in Algorithm 1 and we discovered that this dependency does not affect the performance of the algorithm. I.e., the number of queries computed by the theory that ignores the dependency of the error bits matches the practical results. We need $n = n' + (a-1)2^b$ (and not $n = 2^b n' + (a-1)2^b$) queries in order to recover one block of the secret. The theoretical and practical results are presented in Section 5. Given our practical experiments, we keep the "heuristic" assumption of independence and the algorithm as described in [32] which we called BKW*. Thus, we assume from now on the independence of the noise bits and the independence of the queries.

Another discussion on the independence of the noise bits is presented in [18]. There we can see what is the probability to have a collision, i.e. two queries that share an error bit, among the queries formed during the xoring steps.

We can repeat the algorithm a times, with the same queries, to recover all the k bits. The total time complexity for the reduction phase is $O(ka^2n)$ as we perform the steps described above a times (instead of $O(kan)$ as given in [32]). However, by making the selection of a and b adaptive with ab near to the

[1] Definition 2 of [32] assumes independence of samples but Lemma 2 of [32] shows the reduction without proving independence.

remaining number of bits to recover, we can show that the total complexity is dominated by the one of recovering the first block. So, we can typically concentrate on the algorithm to recover a single block. We provide a more complete analysis in Section 3.

Solving phase. The BKW solving method recovers the 1-bit secret by applying the majority rule. The queries from the reduction phase are of the form $b'_j = s_i \oplus d'_j$, $d'_j \leftarrow Ber_{(1-\delta^{2^{a-1}})/2}$ and s_i being the i^{th} bit of the secret s. Given that the probability for the noise bit to be set to 1 is smaller than $\frac{1}{2}$, in more than half of the cases, these queries will be s_i. Thus, we decide that the value of s_i is given by the majority rule (steps 12-14 of Algorithm 1). By applying the Chernoff bounds [12], we find how many queries are needed such that the probability of guessing incorrectly one bit of the secret is bounded by some constant ε, with $0 < \varepsilon < 1$.

The time complexity of performing the majority rule is linear in the number of queries.

Complexity analysis. With their analysis, Levieil and Fouque [32] obtain the following result:

Theorem 1 (Th. 1 from [32]). *For $k = a \cdot b$, the* BKW* *algorithm heuristically ($n = 20 \cdot \ln(4k) \cdot 2^b \cdot \delta^{-2^a} + (a-1)2^b, t = O(kan), m = kn, \theta = \frac{1}{2}, b$)-solves the* LPN *problem.*[2]

In Section 3 we will see that our theoretical analysis, which we believe to be more intuitive and simpler, gives tighter bounds for the number of queries.

LF1 **Algorithm** During the solving phase, the BKW algorithm recovers the value of the secret bit by bit. Given that we are interested only in queries with Hamming weight 1, many queries are discarded at the end of the reduction phase. As first noted in [32], this can be improved by using a Walsh-Hadamard transform instead of the majority rule. This improvement of BKW is denoted in [32] by LF1. Again, we present the algorithm in pseudo-code in Algorithm 2. As in BKW*, we can concentrate on the complexity to recover the first block.

Reduction phase. The reduction phase for LF1 follows the same steps as in BKW* in obtaining new queries as 2^{a-1} xors of initial queries in order to reduce the secret to size b. At this step, the algorithm does not discard queries anymore but proceeds directly with the solving phase (see steps 3-10 of Algorithm 2). We now have $n' = n - (a-1)2^b$ queries after this phase.

Solving phase. The solving phase consists in applying a Walsh-Hadamard transform in order to recover b bits of the secret at once (steps 11-13 in Algorithm 2). We can recover the b-bit secret by computing the Walsh transform of the function $f(x) = \sum_i 1_{v'_i = x}(-1)^{b'_i}$. The Walsh transform is $\hat{f}(\nu) = \sum_x (-1)^{\nu \cdot x} f(x) = \sum_x (-1)^{\nu \cdot x} \sum_i 1_{v'_i = x}(-1)^{b'_i} = \sum_i (-1)^{(v'_i \cdot \nu) + b'_i} = n' - 2HW(A'\nu + b')$. For $\nu = s$, we have $\hat{f}(s) = n' - 2 \cdot HW(d')$, where d' represents the noise vector after the reduction phase. We know that most of the noise bits are set to 0. So, $\hat{f}(s)$ is large and we suppose it is the largest value in the table of $\hat{f}$. Thus, we have to look at the maximum value of the Walsh transform in order to recover the value of s. A naive implementation of a Walsh transform would give a complexity of 2^{2b} since we apply it on a space of size 2^b. Since we apply a fast Walsh-Hadamard transform, we get a time complexity of $b2^b$ [13].

Complexity analysis. The following theorem states the complexity of LF1:

Theorem 2 (Th. 2 from [32]). *For $k = a \cdot b$ and $a > 1$, the* LF1 *algorithm heuristically ($n = (8b + 200)\delta^{-2^a} + (a-1)2^b, t = O(kan + b2^b), m = kn + b2^b, \theta = \frac{1}{2}, b$)-solves the* LPN *problem.*[3]

[2] The term $(a-1)2^b$ is not included in Theorem 1 from [32]. This factor represents the number of queries lost during the reduction phase and it is the dominant one for all the algorithms except BKW*.

[3] The term $b2^b$ in the time complexity is missing in [32]. While in general kan is the dominant term, in the special case where $a = 1$ (thus we apply no reduction step) a complexity of $O(kan)$ would be wrong since, in this case, we apply the Walsh transform on the whole secret and the term $k2^k$ dominates the final complexity.

Algorithm 2 LF1 Algorithm

Input: a set V of n queries $(v_i, b_i) \in \{0,1\}^{k+1}$ from the LPN oracle, values a, b such that $k = ab$

Output: values $s_1, \dots, s_b$

Partition the positions $\{1, \dots, k\} \setminus \{1, \dots, b\}$ into disjoint $q_1 \cup \dots \cup q_{a-1}$ with q_i of size b

for $i = 1$ to $a-1$ **do** ▷ Reduction phase

  Partition $V = V_1 \cup \dots \cup V_{2^b}$ s.t. vectors in V_j have the the same bit values on q_i

  foreach V_j

    Choose a random $(v^*, b^*) \in V_j$ as a representative vector

    Replace each (v,b) by $(v,b) \oplus (v^*, b^*)$, $(v,b) \in V_j$ for $(v,b) \neq (v^*, b^*)$

    Discard (v^*, b^*) from V_j

  $V = V_1 \cup \dots \cup V_{2^b}$

$f(x) = \sum_{(v,b) \in V} 1_{v_{1,\dots,b} = x} (-1)^{b_i}$ ▷ Solving phase

$\hat{f}(\nu) = \sum_x (-1)^{\nu \cdot x} f(x)$ ▷ Walsh transform of $f(x)$

$(s_1, \dots, s_b) = \mathsf{arg\,max}(\hat{f}(\nu))$

return $s_1, \dots, s_b$

The analysis is similar to the one done for BKW*, except that we now work with blocks of the secret s and not bits. Thus, we bound by $\frac{1}{2a}$ the probability that $\hat{f}(s') > \hat{f}(s)$, where s' is any of the $2^b - 1$ values different from s. As for BKW*, we will provide a more intuitive and tighter analysis for LF1 in Section 3.2.

BKW* *vs.* LF1. We can see that compared to BKW*, LF1 brings a significant improvement in the number of queries needed. As expected, the factor 2^b disappeared as we did not discard any query at the end of the reduction phase. There is an increase in the time and memory complexity because of the fast Walsh-Hadamard transform, but these terms are not the dominant ones.

LF2 **Algorithm** LF2 is a heuristic algorithm, also introduced in [32], that applies the same Walsh-Hadamard transform as LF1, but has a different reduction phase. We provide the pseudocode for LF2 below.

Algorithm 3 LF2 Algorithm

Input: a set V of n queries $(v_i, b_i) \in \{0,1\}^{k+1}$ from the LPN oracle, values a, b such that $k = ab$

Output: values $s_1, \dots, s_b$

Partition the positions $\{1, \dots, k\} \setminus \{1, \dots, b\}$ into disjoint $q_1 \cup \dots \cup q_{a-1}$ with q_i of size b

for $i = 1$ to $a-1$ **do** ▷ Reduction phase

  Partition $V = V_1 \cup \dots \cup V_{2^b}$ s.t. vectors in V_j have the the same bit values on q_i

  foreach V_j

    $V'_j = \emptyset$

    foreach pair $(v,b), (v',b') \in V_j$, $(v,b) \neq (v',b')$

      $V'_i = V'_i \cup (v \oplus v', b \oplus b')$

  $V = V'_1 \cup \dots \cup V'_{2^b}$

$f(x) = \sum_{(v,b) \in V} 1_{v_{1,\dots,b} = x} (-1)^{b_i}$ ▷ Solving phase

$\hat{f}(\nu) = \sum_x (-1)^{\nu \cdot x} f(x)$ ▷ compute the Walsh transform of $f(x)$

$(s_1, \dots, s_b) = \mathsf{arg\,max}(\hat{f}(\nu))$

return $s_1, \dots, s_b$

Reduction phase. Similarly to BKW* and LF1, the n queries are grouped into equivalence classes. Two queries are in the same equivalence class if they have the same value on a window of b bits. In each equivalence class we perform the xor of all the pairs from that class. Thus, we do not choose any representative vector that is discarded afterwards. Given that in an equivalence class there are $n/2^b$ queries, we expect to have $2^b\binom{n/2^b}{2}$ queries at the end of the xor-ing. One interesting case is when n is of the form $n = 3\cdot 2^b$ as with this reduction phase we expect to preserve the number of queries since $\binom{3}{2} = 3$. For any $n > 3\cdot 2^b$, the number of queries will grow exponentially and will also affect the time and memory complexity.

Solving phase. This works like in LF1.

In a scenario where the attacker has access to a restricted number of queries, this heuristic algorithm helps in increasing the number of queries. With LF2, the attacker might produce enough queries to recover the secret value s.

FMICM **Algorithm** Another algorithm by Fossorier et al. [19] uses ideas from fast correlation attacks to solve the LPN problem. While there is an improvement compared with the BKW* algorithm, this algorithm does not perform better than LF1 and LF2. Given that it does not bring better results, we just present the main steps of the algorithm.

As the previous algorithms, it can be split into two phases: reduction and solving phase. The reduction phase first decimates the number of queries and keeps only those queries that have 0 bits on a window of a given size. Then, it performs xors of several queries in order to further reduce the size of the secret. The algorithm that is used for this step is similar to the one that constructs parity checks of a given weight in correlation attacks. The solving phase makes use of the fast Walsh-Hadamard transform to recover part of the secret. By iteration the whole secret is recovered.

Covering Codes Algorithm The new algorithm [23] that was presented at Asiacrypt'14, introduces a new type of reduction. There is a difference between [23] and what was presented at the Asiacrypt conference (mostly due to our results). We concentrate here on [23] and in the next section we present the suggestions we provided to the authors.

Reduction phase. The first step of this algorithm is to transform the LPN instance where the secret s is randomly chosen to an instance where the secret has now a Bernoulli distribution. This method was described in [31,4].

Given n queries from the LPN oracle: $(\overline{v_1}, b_1)$, $(\overline{v_2}, b_2), \dots, (\overline{v_n}, b_n)$, select k linearly independent vectors $\overline{v}_{i_1}, \dots, \overline{v}_{i_k}$. Construct the $k \times k$ target matrix M that has on its columns the aforementioned vectors, i.e. $M = [\overline{v}_{i_1}^T \overline{v}_{i_2}^T \dots \overline{v}_{i_k}^T]$. Compute $(M^T)^{-1}$ the inverse of M^T, where M^T is the transpose of M. We can rewrite the k queries corresponding to the selected vectors as $M^T s + d'$, where d' is the k-bit vector $d = (d_{i_1}, d_{i_2}, \dots, d_{i_k})$. We denote $b' = M^T s + d'$. For any $\overline{v_j}$ that is not used in matrix M do the following computation:

$$\overline{v}_j (M^T)^{-1} b' + b_j = \langle \overline{v}_j (M^T)^{-1}, d' \rangle + d_j.$$

From the initial set of queries, we have obtained a new set where the secret value is d'. This can be seen as a reduction to a sparse secret. The complexity of this transform is $O(k^3 + nk^2)$ by the schoolbook matrix inversion algorithm. This can be improved as follows: for a fixed s, one can split the matrix $(M^T)^{-1}$ in $a = \lceil \frac{k}{s} \rceil$ parts $\begin{bmatrix} M_1 \\ M_2 \\ \dots \\ M_a \end{bmatrix}$ of s rows. By pre-computing $\overline{v} M_i$ for all $\overline{v} \in \{0,1\}^s$, the operation of performing $\overline{v}_j (M^T)^{-1}$ takes $O(ka)$. The pre-computation takes $O(2^s)$ and is negligible if the memory required by the BKW reduction is bigger. With this pre-computation the complexity is $O(nka)$.

Afterwards the algorithm follows the usual BKW reduction steps where the size of the secret is reduced to k' by the xoring operation. Again the vector of k bits is seen as being split into blocks of size b. The BKW reduction is applied t times. Thus, we have $k' = k - tb$.

The secret s of k' bits is split into 2 parts: one part denoted s_1 of k'' bits and the other part, denoted s_2, of $k' - k''$ bits. The next step in the reduction is to guess value of s_2 by making an assumption on its Hamming weight: $HW(s_2) \leq w_0$. The remaining queries are of the form $(v_i, b_i = \langle v_i, s_2 \rangle \oplus d_i)$, where $v_i, s_2 \in \{0,1\}^{k''}$ and $d_i \in Ber_{\frac{1-\delta^{2t}}{2}}$. Thus, the problem is reduced to a secret of k'' bits.

At this moment, the algorithm approximates the v_i vectors to the nearest codeword g_i in a $[k'', \ell]$-code where k'' is the size and ℓ is the dimension. By observing that g_i can be written as $g_i = g_i' G$, where G is the generating matrix of the code, we can write the equations in the form

$$b_i = \langle v_i, s_2 \rangle \oplus d_i = \langle g_i' G, s_2 \rangle \oplus \langle v_i - g_i, s_2 \rangle \oplus d_i = \langle g_i', s_2' \rangle \oplus d_i'$$

with $s_2' = G s_2$ and $d_i' = \langle v_i - g_i, s_2 \rangle \oplus d_i$, where g_i', s_2' have length ℓ. If the code has optimal covering radius d, $v_i - g_i$ is a random vector of weight bounded by d, while s_2 is a vector of some small weight bounded by c, with some probability. So, $\langle v_i - g_i, s' \rangle$ is biased and we can treat d_i' in place of d_i.

In [23], the authors approximate the bias of $\langle v_i - g_i, s_2 \rangle$ to $\delta' = \left(1 - 2\frac{d}{k''}\right)^c$, as if all bits were independent. As discussed in the next section, this approximation is far from good.

No queries are lost during this covering code operation and now the secret is reduced to ℓ bits. We now have $n' = n - k - t2^b$ queries after this phase.

Solving phase. The solving phase of this algorithm follows the same steps as LF1, i.e. it employs a fast Walsh-Hadamard transform. One should notice that the solving phase recovers ℓ relations between the bits of the secret and not actual ℓ bits of the secret.

Complexity analysis. Recall that in the algorithm two assumptions are made regarding the Hamming weight of the secret: that s_2 has a Hamming weight smaller than c and that s_1 has a Hamming weight smaller than w_0. This holds with probability $\Pr(w_0, k' - k'') \cdot \Pr(c, k'')$ where

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

The total complexity is given by the complexity of one iteration to which we add the number of times we have to repeat the iteration. We state below the result from [23]:

Theorem 3 (Th 1. from [23]).
Let n be the number of samples required and $t, a, b, w_0, c, l, k', k''$ be the algorithm parameters. For the $\mathsf{LPN}_{k,\tau}$ instance, the number of bit operations required for a successful run of the new attack is equal to

$$C^* = \frac{C_{\textit{sparse reduction}} + C_{\textit{bkw reduction}} + C_{\textit{guess}} + C_{\textit{covering code}} + C_{\textit{Walsh transform}}}{\Pr(w_0, k' - k'')\Pr(c, k'')},$$

where

- $C_{\textit{sparse reduction}} = nka$ *is the cost of reducing the* LPN *instance to a sparse secret*
- $C_{\textit{bkw reduction}} = (k+1)tn$ *is the cost of the* BKW *reduction steps*
- $C_{\textit{guess}} = n' \sum_{i=0}^{w_0} \binom{k'-k''}{i} i$ *is the cost of guessing $k' - k''$ bits and $n' = n - k - t2^b$ represents the number of queries at the end of the reduction phase*
- $C_{\textit{covering code}} = (k'' - \ell)(2n' + 2^{\ell})$ *is the cost of the covering code reduction and n' is again the number of queries*
- $C_{\textit{Walsh transform}} = \ell 2^{\ell} \sum_{i=0}^{w_0} \binom{k'-k''}{i}$ *is the cost of applying the fast Walsh-Hadamard transform for every guess of $k' - k''$ bits*

under the condition that $n - t2^b > \frac{1}{\delta^{2^{t+1}} \cdot \delta'^2}$, *where* $\delta = 1 - 2\tau$ *and* $\delta' = \left(1 - 2\frac{d}{k''}\right)^c$ *and* d *is the smallest integer, s.t.* $\sum_{i=0}^{d} \binom{k''}{i} > 2^{k''-\ell}$.

The condition $n - t2^b > \frac{1}{\delta^{2^{t+1}} \cdot \delta'^2}$ proposed in [23] imposes a lower bound on the number of queries needed in the solving phase for the fast Walsh-Hadamard transform. In our analysis, we will see that this is underestimated: the Chernoff bounds dictate a larger number of queries.

3 Tighter Theoretical Analysis

In this section we present a different theoretical analysis from the one of Levieil and Fouque [32] for the solving phases of the LPN solving algorithms. A complete comparison is given in Section 5. Our analysis gives tighter bounds and aims at closing the gap between theory and practice. For the new algorithm from [23], we present the main points that we found to be incomplete.

We first show how the cost of solving one block of the secret dominates the total cost of recovering s. The main intuition is that after recovering a first block of k' secret bits, we can apply a simple back substitution mechanism and consider solving a $\mathsf{LPN}_{k-k',\tau}$ problem. The same strategy is applied by [2] when solving LWE. Note that this is simply a generalisation of the classic Gaussian elimination procedure for solving linear systems, where we work over blocks of bits.

Specifically, let $k_1 = k$ and $k_i = k_{i-1} - k'_{i-1}$ for $i > 1$. Now, suppose we were able to $(n_i, t_i, m_i, \theta_i, k'_i)$-solve an $\mathsf{LPN}_{k_i,\tau}$ instance (meaning we recover a block of size k'_i from the secret of size k_i with probability θ_i, in time t_i and with memory m_i). One can see that for $k_{i+1} < k_i$ we need less queries to solve the new instance (the number of queries is dependent on the size k_{i+1} and on the noise level). With a smaller secret, the time complexity will decrease. Having a shorter secret and less queries, the memory needed is also smaller. Then, we can (n, t, m, θ, k)-solve the problem $\mathsf{LPN}_{k,\tau}$ (i.e recover s completely), with $n = \max(n_1, n_2, \ldots)$, $\theta = \theta_1 + \theta_2 + \ldots$, $t = t_1 + k'_1 n_1 + t_2 + k'_2 n_2 \ldots$ (the terms $k'_i n_i$ are due to query updates by back substitution) and $m = \max(m_1, m_2, \ldots)$. Finally, by taking $\theta_i = 3^{-i}$, we obtain $\theta \leq \frac{1}{2}$ and thus recover the full secret s with probability over 50%.

It is easily verified that for all the algorithms we consider, we have $n = n_1$, $m = m_1$, and t is dominated by t_1. We provide an example on a concrete LPN instance in Appendix B.

For all the solving algorithms presented in this section we assume that n' queries remain after the reduction phase and that the bias is δ'. For the solving techniques that recover the secret block-by-block, we assume the block size to be k'.

3.1 BKW^* Algorithm

Given an LPN instance, the BKW^* solving method recovers the 1 bit secret by applying the majority rule. Recall that the queries are of the form $b'_j = s_i \oplus d'_j$, $d'_j \leftarrow Ber_{(1-\delta')/2}$. The majority of these queries will most likely be $b'_j = s_i$. It is intuitive to see that the majority rule fails when more than half of the noise bits are 1 for a given bit. Any wrong guess of a bit gives a wrong value of the k-bit secret s. In order to bound the probability of such a scenario, we use the Hoeffding bounds [25] with $X_j = e_j$ (See Appendix A). We have $\Pr[X_j = 1] = \frac{1-\delta'}{2}$. For $X = \sum_{j=1}^{n'} X_j$, we have $E(X) = \frac{(1-\delta')n'}{2}$ and we apply Theorem 12 with $t = \frac{\delta n'}{2}$, $\alpha_j = 0$ and $\beta_j = 1$ and we obtain

$$\Pr[\text{incorrect guess on } s_i] = \Pr\left[X \geq \frac{n'}{2}\right] \leq e^{-\frac{n'\delta'^2}{2}}.$$

As discussed in Remark 1, the assumption of independence is heuristic.

Using the above results for every bit $1, \ldots, b$, we can bound by a constant θ, the probability that we guess incorrectly a block of s, with $0 < \theta < 1$. Using the union bound, we get that $n' = 2\delta'^{-2} \ln(\frac{b}{\theta})$. Given that $n' = \frac{n-(a-1)2^b}{2^b}$ and that $\delta' = \delta^{2^{a-1}}$, we obtain the following result.

Theorem 4. *For $k \leq a \cdot b$, the* BKW* *algorithm heuristically ($n = 2^{b+1}\delta^{-2^a}\ln(\frac{b}{\theta}) + (a-1)2^b, t = O(kan)$, $m = kn, \theta, b$)-solves the* LPN *problem.*

We note that we obtained the above result using the union bound. One could make use of the independence of the noise bits and obtain $n = 2^{b+1}\delta^{-2^a}\ln\left(\frac{1}{1-2^{-1/k}}\right) + (a-1)2^b$, but this would bring a very small improvement.

In terms of query complexity, we compare our theoretical results with the ones from [32] in Table 1 and Table 2. We provide the $\log_2(n)$ values for k varying from 32 to 100 and we take different Bernoulli noise parameters that vary from 0.01 to 0.4. Overall, our theoretical results bring an improvement of a factor 10 over the results of [32].

τ	k				
	32	48	64	80	100
0.01	10.97	12.82	15.93	18.66	21.74
0.10	15.84	20.01	24.12	28.20	33.28
0.20	19.71	24.85	30.97	34.83	39.90
0.25	21.81	26.95	33.07	38.14	44.11
0.40	28.24	36.38	43.64	48.71	55.78

Table 1: BKW* query complexity - our theory

τ	k				
	32	48	64	80	100
0.01	14.56	16.60	19.68	22.59	25.64
0.10	19.75	23.87	27.95	32.00	37.06
0.20	23.50	28.61	34.69	38.64	43.70
0.25	25.60	30.72	36.79	41.85	47.90
0.40	31.89	40.00	47.37	52.43	59.48

Table 2: BKW* query complexity - theory from [32]

In Section 5.1 we show that Theorem 4 gives results that are very close to the ones we measure experimentally.

We note that our BKW* algorithm, for which we have stated the above theorem, follows the steps from Algorithm 1 for $k = a \cdot b$. For $k < a \cdot b$ the algorithm is a bit different. In this case we have $a-1$ blocks of size b and an incomplete block of size smaller than b. During the reduction phase, we first partition the incomplete block and then apply $(a-2)$ reduction steps for the complete blocks. We finally have b bits to recover. Other than this small change, the algorithm remains the same.

If the term $2^{b+1}\delta^{-2^a}\ln(\frac{b}{\theta_i})$ dominates n, the next iteration can use a decreased by 1 leading to a new $n \approx 2^{b+1}\delta^{-2^{a-1}}\ln(\frac{b}{\theta_{i+1}})$ which is roughly the square root of the previous n. So, the complexity of recovering this block is clearly dominated by the cost of recovering the previous block. If the term $(a-1)2^b$ is dominating, we can decrease b by one in the next block and reach the same conclusion.

3.2 LF1 **Algorithm**

For the LF1 algorithm, the secret is recovered by choosing the highest value of a Walsh-Hadamard transform. Recall that the Walsh transform is $\hat{f}(\nu) = n' - 2HW(A'\nu + b')$. For $\nu = s$, we obtain that the Walsh transform has the value $\hat{f}(s) = n' - 2HW(d')$. We have $E(\hat{f}(s)) = n'\delta'$.

The failure probability for LF1 is bounded by the probability that there is another vector $\nu \neq s$ such that $HW(A'\nu + b') \leq HW(A's + b')$. Recall that $A's + b' = d'$. We define $x = s + \nu$ so that $A'\nu + b' = A'x + d'$. We obtain that the failure probability is bounded by $2^{k'}$ times the probability that $HW(A'x + d') \leq HW(d')$, for a fixed k'-bit non-zero vector x. As A' is uniformly distributed, independent from d', and x is fixed and non-zero, $A'x + d'$ is uniformly distributed, so we can rewrite the inequality as $HW(y) \leq HW(d')$, for a random y.

To bound the failure probability, we again use the Hoeffding inequality [25]. Let $X_1, X_2, \ldots, X_{n'}$ be random independent variables with $X_j = y_j - d'_j$, $\Pr(X_j \in [-1,1]) = 1$. We have $E(y_j - d'_j) = \frac{\delta'}{2}$. We can take $t = E[X] = \frac{\delta' n'}{2}$ in Theorem 12 and obtain:

$$\Pr[\text{incorrect guess on one block}] \leq 2^{k'} \Pr\left[\sum_{j=1}^{n'} (y_j - d'_j) \leq 0\right] \leq 2^{k'} e^{-\frac{n'\delta'^2}{8}}.$$

Again we can bound the probability of incorrectly guessing one block of s by θ. With $n' = 8(\ln \frac{2^{k'}}{\theta})\delta'^{-2}$, the probability of failure is smaller than θ. The total number of queries will be $n = n' + (a-1)2^b$, we have $\delta' = \delta^{2^{a-1}}$ and $k' = b$. Similar to BKW, we obtain the following theorem:

Theorem 5. *For $k \leq a \cdot b$, the* LF1 *algorithm heuristically ($n = 8\ln(\frac{2^b}{\theta})\delta^{-2^a} + (a-1)2^b, t = O(kan + b2^b), m = kn + b2^b, \theta, b$)-solves the* LPN *problem.*

By comparing the term $(8b+200)\delta^{-2^a}$ in Theorem 2 with our value of $8\ln(\frac{2^b}{\theta})\delta^{-2^a}$, one might check that our term is roughly a factor 2 smaller than that of [32] for practical values of a and b. For example, for a $\mathsf{LPN}_{768,0.01}$ instance (with $a = 11$, $b = 70$), our analysis requires 2^{68} queries for the solving phase while the Levieil and Fouque analysis requires 2^{69} queries.

3.3 LF2 algorithm

Having the new bounds for LF1, we can state a similar result for LF2. Recall that when $n = 3 \cdot 2^b$, LF2 preserves the number of queries during the reduction phase. For $3 \cdot 2^b \geq n'$ we have that:

Theorem 6. *For $k \leq a \cdot b$ and $n = 3 \cdot 2^b \geq 8\ln(\frac{2^b}{\theta})\delta^{-2^a}$, the* LF2 *algorithm heuristically ($n = 3 \cdot 2^b, t = O(kan + b2^b), m = kn + b2^b, \theta, b$)-solves the* LPN *problem.*

One can observe that we may allow for n to be smaller than $3 \cdot 2^b$. Given that the solving phase may require less than $3 \cdot 2^b$, we could start with less queries, decrease the number of queries during the reduction and end up with the exact number of queries needed for the solving phase.

3.4 Covering Codes Algorithm

Recall that the algorithm first reduces the size of the secret to k'' bits by running BKW reduction steps. Then it approximates the v_i vector to the nearest codeword g_i in a $[k'', \ell]$-code with G as generator matrix. The noisy inner products can be rewritten as

$$b_i = \langle g'_i G, s_2 \rangle \oplus \langle v_i - g_i, s_2 \rangle \oplus d_i = \langle g'_i, G^T s_2 \rangle \oplus d'_i = \langle g'_i, s'_2 \rangle \oplus d'_i,$$

where $g_i = g'_i G$, $s'_2 = G^T s_2$ and $d'_i = \langle g_i - v_i, s_2 \rangle \oplus d_i$.

Given that the code has a covering radius of d and that the Hamming weight of s_2 is smaller than c, the bias of $\langle g_i - v_i, s_2 \rangle$ is computed as $\delta' = \left(1 - 2\frac{d}{k''}\right)^c$ in [23], where k'' is the size of s_2. We stress that this approximation is far from good.

Indeed, with the $[3,1,3]$ repetition code given as an example in [23], the xor of two error bits is unbiased. Even worse: the xor of the three bits has a negative bias. So, when using the code obtained by 25 concatenations of this repetition code and $c = 6$, with some probability of 36% we have at least two error bits falling in the same concatenation and the bias makes this approach fail.

We can do the same computation with the concatenation of five $[23,12]$ Golay codes with $c = 15$, as suggested in [23]. With probability 0.21%, the bias is zero or negative so the algorithm fails. With some probability 8.3%, the bias is too low.

In any case, we cannot take the error bits as independent. When the code has optimal covering radius, we can actually find an explicit formula for the bias of $\langle v_i - g_i, s_2\rangle$ assuming that s_2 has weight c:

$$\Pr[\langle v_i - g_i, s_2\rangle = 1 | HW(s_2) = c] = \frac{1}{S(k'', d)} \sum_{i \le d, i \text{ odd}} \binom{c}{i} S(k'' - c, d - i)$$

where $S(k'', d)$ is the number of k''-bit strings with weight at most d.

To solve $\mathsf{LPN}_{512,0.125}$, [23] proposes the following parameters

$$t = 6 \quad a = 9 \quad b = 63 \quad \ell = 64 \quad k'' = 124 \quad w_0 = 2 \quad c = 16$$

and obtain $n = 2^{66.3}$ and a complexity of $2^{79.92}$. With these parameters, [23] approximated the bias to $\left(1 - 2\frac{d}{k''}\right)^c = 2^{-5.91}$ (with $d = 14$). With our exact formula, the bias should rather be of $2^{-7.05}$. So, n should be multiplied by 4.82 (the square of the ratio).

Also, we stress that all this assumes the construction of a code with optimal radius coverage. One example is the Golay codes. But this code can be used only for few LPN instances. If we use concatenations of repetition codes, given as an example in [23], the formula for the bias changes. Given ℓ concatenations of the $[k_i, 1]$ repetition code, with $k_1 + \ldots + k_\ell = k''$, $k_i \approx \frac{k''}{\ell}$ and $1 \le i \le \ell$, we would have to split the secret s_2 in chunks of $k_1, \ldots, k_\ell$ bits. We take $c_1 + \ldots + c_\ell = c$ where c_i is the weight of s_2 on the i^{th} chunk. In this case the bias for each repetition code is

$$\delta_i = 1 - 2 \times \frac{1}{S(k_i, d_i)} \sum_{j \le d_i, j \text{ odd}} \binom{c_i}{j} S(k_i - c_i, d_i - j),$$

where $d_i = \lfloor \frac{k_i}{2} \rfloor$.

The final bias is $\delta' = \delta_1 \cdots \delta_\ell$.

We emphasize that the value of n is underestimated in [23]. Indeed, with $n' = \mathsf{bias}^{-2}$, the probability that $\arg\max(\hat{f}(\nu)) = s_2'$ is too low in $\mathsf{LF1}$. To have a constant probability of success θ, our analysis says that we should multiply n' by $8\ln(\frac{2^\ell}{\theta})$. For $\mathsf{LPN}_{512,0.125}$ and $\theta = \frac{1}{3}$, this is 363.

When presenting their algorithm at Asiacrypt'14, the authors of [23] updated their computation by using our suggested formulas for the bias and the number of queries. In order to obtain a complexity smaller than 2^{80}, they further improved their algorithm by the following observation: instead of assuming that the secret s_2 has a Hamming weight smaller or equal to c, the algorithm takes now into account all the Hamming weights that would give a good bias for the covering code reduction. I.e., the algorithm takes into account all the Hamming weights c_i for which $\delta' > \varepsilon_{\mathsf{set}}$, where $\varepsilon_{\mathsf{set}}$ is a preset bias. The probability of a good secret changes from $\Pr(c, k'')$ to $\Pr(HW)$ that we define below. They further adapted the algorithm by using the $\mathsf{LF2}$ reduction steps. Recall that for $n = 3 \cdot 2^b$, the number of queries are preserved during the reduction phase. With these changes they propose the following parameters for $\mathsf{LPN}_{512,0.125}$:

$$t = 5 \quad b = 62 \quad \ell = 60 \quad k'' = 180 \quad w_0 = 2 \quad \varepsilon_{\mathsf{set}} = 2^{-14.18}$$

Using two $[90, 30]$ codes, they obtain that $n = 2^{63.6} = 3 \cdot 2^b$ queries are needed, the memory used is of $m = 2^{72.6}$ bits and the time complexity is $C^* = 2^{79.7}$. Thus, this algorithm gives better performance than $\mathsf{LF2}$ and shows that this LPN instance does not offer a security of 80 bits. [4]

With all the above observations we update the Theorem 3.

Theorem 7. *The covering code (*$n = 8\ln(\frac{2^\ell}{\theta})\frac{1}{\delta^{2^t}\varepsilon_{\mathsf{set}}^2} + t2^b$ *,*$C^*, m = kn + 2^{k''-\ell} + \ell 2^\ell, \theta, \ell$*)-solves the* LPN *problem* [5]*, where* $\delta = 1 - 2\tau$ *and* $\delta' > \varepsilon_{\mathsf{set}}$ *is the bias introduced by the covering code reduction that is*

[4] For the computation of n the authors of [23] use the term $4\ln(2^\ell)$ instead of $8\ln(\frac{2^\ell}{\theta})$. If we use our formula, we obtain that we need more than $3 \cdot 2^b$ queries and obtain a complexity of $2^{80.08}$.

[5] This n corresponds to covering code reduction using $\mathsf{LF1}$. For $\mathsf{LF2}$ reduction steps we need to have $n = 3 \cdot 2^b + k \ge 8\ln(\frac{2^\ell}{\theta})\frac{1}{\delta^{2^t}\varepsilon_{\mathsf{set}}^2}$.

lower bounded by a preset bias. The code chosen for the covering code reduction step can be expressed as the concatenation of one or more linear codes. The time C^* complexity can be expressed as

$$C^* = \frac{C_{\textsf{sparse reduction}} + C_{\textsf{bkw reduction}} + C_{\textsf{guess}} + C_{\textsf{covering code}} + C_{\textsf{Walsh transform}}}{\Pr(w_0, k' - k'')\Pr(HW)},$$

where

- $C_{\textsf{sparse reduction}} = nka$ *is the cost of reducing the* LPN *instance to a sparse secret*
- $C_{\textsf{bkw reduction}} = (k+1)tn$ *is the cost of the* BKW *reduction steps*
- $C_{\textsf{guess}} = n' \sum_{i=0}^{w_0} \binom{k'-k''}{i} i$ *is the cost of guessing $k' - k''$ bits and $n' = n - k - t2^b$ represents the number of queries at the end of the reduction phase*
- $C_{\textsf{covering code}} = (k'' - \ell)(2n' + 2^{\ell})$ *is the cost of the covering code reduction and n' is again the number of queries*
- $C_{\textsf{Walsh transform}} = \ell 2^{\ell} \sum_{i=0}^{w_0} \binom{k'-k''}{i}$ *is the cost of applying the fast Walsh-Hadamard transform for every guess of $k' - k''$ bits*
- $\Pr(HW) = \sum_{c_i} (1-\tau)^{k''-c_i} \tau^{c_i} \binom{k''}{c_i}$ *where c_i is chosen such that the bias δ', which depends on c_i and the covering radius d of the chosen code, is larger than $\varepsilon_{\textsf{set}}$*

4 Other LPN Solving Algorithms

Most LPN-based encryption schemes use τ as a function of k, e.g. $\tau = \frac{1}{\sqrt{k}}$ [3,14]. The bigger the value of k, the lower the level of noise. For $k = 768$, we have $\tau \approx 0.036$. For such a value we say that the noise is sparse. Given that these LPN instances are used in practice, we consider how we can construct other algorithms that take advantage of this extra information.

The first two algorithms presented in this section bring new ideas for the solving phase. The third one provides a method to recover the whole secret and does not need any reduction phase.

We maintain the notations used in the previous section: n' queries remain after the reduction phase, the bias is δ' and the block size is k'.

For these solving algorithms, we assume that the secret is sparse. Even if the secret is not sparse, we can just assume that the noise is sparse. We can transform an LPN instance to an instance of LPN where the secret is actually a vector of noise bits by the method presented in [31]. The details of this transform were given in Section 2.2 for the covering codes algorithm.

We denote by Δ the sparseness of the secret, i.e. $\Pr[s_i = 1] = \frac{1-\Delta}{2}$ for any $1 \leq i \leq k$. We say that the secret is Δ-sparse. Given the transformation explained above, we can take $\Delta = \delta$.

The assumption we make is that the Hamming weight of the k'-bit length secret s is in a given range. On average we have that $HW(s) = k'(\frac{1-\Delta}{2})$, so an appropriate range is $\left[0, k'(\frac{1-\Delta}{2}) + \frac{\sigma}{2}\sqrt{k'}\right]$, where σ is constant. We denote $k'(\frac{1-\Delta}{2})$ by E_{HW} and $\frac{\sigma}{2}\sqrt{k'}$ by $\textsf{dev}$. Thus, we are searching in the range $[0, E_{HW} + \textsf{dev}]$. We can bound the probability that the secret has a Hamming weight outside the range by using the Hoeffding bound [25].

Let $X_1, X_2, \ldots, X_{k'}$ be independent random variables that correspond to the secret bits, i.e. $\Pr[X_i = 1] = \frac{1-\Delta}{2}$ and $\Pr(X_i \in [0,1]) = 1$. We have $E(X) = \frac{1-\Delta}{2}k'$. Using Theorem 12, we get that

$$\Pr[HW(s) \textsf{ not in range}] = \Pr\left[HW(s) - \frac{(1-\Delta)}{2}k' \geq \sigma\sqrt{\frac{k'}{4}}\right] \leq e^{-\frac{\sigma^2}{2}}.$$

If we want to bound by $\theta/2$ the probability that $HW(s)$ is not in the correct range for one block, we obtain that $\sigma = \sqrt{2\ln(\frac{2}{\theta})}$.

4.1 Exhaustive search on sparse secret

We have $S = \sum_{i=0}^{E_{HW}+\mathsf{dev}} \binom{k'}{i}$ vectors ν with Hamming weight in our range. One first idea would be to perform an exhaustive search on the sparse secret. We denote this algorithm by Search_1. For every such value ν, we compute $HW(A\nu + b)$. In order to compute the Hamming weight we have to compute the multiplication between A and all ν which have the Hamming weight in the correct range. This operation would take $O(Sn'k')$ time but we can save a k' factor by the following observation done in [7]: computing $A\nu$, with $HW(\nu) = i$ means xoring i columns of A. If we have the values of $A\nu$ for all ν where $HW(\nu) = i$ then we can compute $A\nu'$ for $HW(\nu') = i+1$ by adding one extra column to the previous results.

We use here a similar reasoning done for the Walsh-Hadamard transform. When $\nu = s$, the value of $HW(As+b)$ is equal to $HW(d)$ and we assume that this is the smallest value as we have more noise bits set on 0 than 1. Thus, going through all possible values of ν and keeping the minimum will give us the value of the secret. The time complexity of Search_1 is the complexity of computing the Hamming weight, i.e. $O(Sn')$.

Besides Search_1, which requires a matrix multiplication for each trial, we also discovered that a Walsh transform can be used for a sparse secret. We call this algorithm Search_2. The advantage is that a Walsh transform is faster than a naive exhaustive search and thus improves the time complexity. We thus compute the fast Walsh-Hadamard transform and search the maximum of $\hat{f}$ only for those S values with Hamming weight in the correct range. Given that we apply a Walsh transform we get that the complexity of this solving algorithm is $O(k'2^{k'})$. So, it is more interesting than Search_1 when $Sn' > k'2^{k'}$.

For both algorithms the failure probability is given by the scenario where there exists another sparse value $\nu \neq s$ such that $HW(A\nu+b) \leq HW(As+b)$. As we search through S possible values for the secret we obtain that

$$\Pr[\mathsf{incorrect\ guess\ on\ one\ block}] \leq Se^{-\frac{n'\delta'^2}{8}}.$$

The above probability accounts for only one block of the secret. Thus we can say that with $\sigma = \sqrt{2\ln(\frac{2}{\theta})}$ and $n = 8(\ln\frac{2S}{\theta})\delta^{-2^a} + (a-1)2^b$, the probability of failure is smaller than θ.

Another failure scenario, that we take into account into our analysis, occurs when the secret has a Hamming weight outside our range.

Complexity analysis. Taking $n = n' + (a-1)2^b$, $k' = b$, $\delta' = \delta^{2^{a-1}}$ and $\Delta = \delta$, we obtain the following theorems for Search_1 and Search_2:

Theorem 8. *Let $S = \sum_{i=0}^{E_{HW}+\mathsf{dev}} \binom{b}{i}$ where $E_{HW} = b(\frac{1-\Delta}{2})$ and $\mathsf{dev} = \frac{\sigma}{2}\sqrt{b}$ and let $n' = 8\ln(\frac{2S}{\theta})\delta^{-2^a}$. For $k \leq a\cdot b$ and a secret s that is Δ-sparse, the Search_1 algorithm heuristically ($n = 8\ln(\frac{2S}{\theta})\delta^{-2^a} + (a-1)2^b, t = O(kan + n'S), m = kn + b\binom{b}{E_{HW}+\mathsf{dev}}, \theta, b$)-solves the LPN problem.*

Theorem 9. *Let $S = \sum_{i=0}^{E_{HW}+\mathsf{dev}} \binom{b}{i}$ where $E_{HW} = b(\frac{1-\Delta}{2})$ and $\mathsf{dev} = \frac{\sigma}{2}\sqrt{b}$. For $k \leq a\cdot b$ and a secret s that is Δ-sparse, the Search_2 algorithm heuristically ($n = 8\ln(\frac{2S}{\theta})\delta^{-2^a} + (a-1)2^b, t = O(kan + b2^b), m = kn, \theta, b$)-solves the LPN problem.*

Here, we take the probability, that any of the two failure scenarios to happen, to be each $\theta/2$. A search for the optimal values such that their sum is θ, brings a very little improvement to our results. Taking $k' = b$, we stress that S is much smaller than the $2^{k'} = 2^b$ term that is used for $\mathsf{LF1}$. For example, for $k = 768$, $a = 11$, $b = 70$ and $\tau = 0.05$, we have that $S \approx 2^{33}$ which is smaller than $2^b = 2^{70}$ and we get $n' = 2^{67.33}$ and $n = 2^{73.34}$ (compared to $n' = 2^{68.32}$ and $n = 2^{73.37}$ for $\mathsf{LF1}$). We thus expect to require less queries for exhaustive search compared to $\mathsf{LF1}$. As the asymptotic time complexity of Search_2 is the same as $\mathsf{LF1}$ and the number of queries is smaller, we expect to see that this algorithm runs faster than $\mathsf{LF1}$.

4.2 Meet in the middle on sparse secret (MITM)

Given that $As+d=b$, we split s into s_1 and s_2 and rewrite the equation as $A_1s_1+d=A_2s_2+b$. With this split, we try to construct a meet-in-the-middle attack by looking for A_2s_2+b close to A_1s_1. The secret s has size k' and we split it into s_1 of size k_1 and s_2 of size k_2 such that $k_1+k_2=k'$. We consider that both s_1 and s_2 are sparse. Thus the Hamming weight of s_i lies in the range $\left[0, k_i(\frac{1-\Delta}{2})+\frac{\sigma'}{2}\sqrt{k_i}\right]$. We denote $k_i(\frac{1-\Delta}{2})+\frac{\sigma'}{2}\sqrt{k_i}$ by $\mathsf{max}_{\mathsf{HW}}(k_i)$. In order to bound the probability that both estimates are correct we use the same bound shown in Section 4 and obtain that $\sigma'=\sqrt{2\ln(\frac{4}{\theta})}$.

For our MITM attack we have a pre-computation phase. We compute and store A_1s_1 for all $S_1=\sum_{i=0}^{\mathsf{max}_{\mathsf{HW}}(k_1)}\binom{k_1}{i}$ possible values for s_1. We do the same for s_2, i.e compute A_2s_2+b for all $S_2=\sum_{i=0}^{\mathsf{max}_{\mathsf{HW}}(k_2)}\binom{k_2}{i}$ vectors s_2. The pre-computation phase takes $(S_1+S_2)n'$ steps in total. Afterwards we pick ξ bit positions and hope that the noise d has only values of 0 on these positions. If this is true, then we could build a mask μ that has Hamming weight ξ such that $d\wedge\mu=0$. The probability for this to happen is $(\frac{1+\delta'}{2})^{\xi}=e^{-\xi\ln\frac{2}{1+\delta'}}$.

We build our meet-in-the-middle attack by constructing a hash table where we store, for all s_2 values, A_2s_2+b at position $h((A_2s_2+b)\wedge\mu)$. We have S_2 vectors s_2, so we expect to have $S_22^{-\xi}$ vectors on each position of the hash table. For all S_1 values of s_1, we check for collisions, i.e. $h((A_1s_1)\wedge\mu)=h((A_2s_2+b)\wedge\mu)$. If this happens, we check if A_1s_1 xored with A_2s_2+b gives a vector d with a small Hamming weight. Remember that with the pre-computed values we can compute d with only one xor operation. If the resulting vector has a Hamming weight in our range, then we believe we have found the correct s_1 and s_2 values and we can recover the value of s. Given that $A_1s_1+A_2s_2+d=b$, we expect to have $(A_2s_2+b)\wedge\mu=A_1s_1\wedge\mu$ only when $d\wedge\mu=0$. The condition $d\wedge\mu=0$ holds with a probability of $(\frac{1+\delta'}{2})^{\xi}$ so we have to repeat our algorithm $(\frac{2}{1+\delta'})^{\xi}$ times in order to be sure that our condition is fulfilled.

As for exhaustive search, we have two scenarios that could result in a failure. One scenario is when s_1 or s_2 have a Hamming weight outside the range. The second one happens when there is another vector $\nu\neq s$ such that $HW(A_1\nu_1+A_2\nu_2+b)\leq HW(A_1s_1+A_2s_2+b)$ and $(A_1\nu_1+A_2\nu_2+b)\wedge\mu=0$. This occurs with probability smaller than $S_1S_2e^{-\frac{n'\delta'^2}{8}}$.

Complexity analysis. The time complexity of constructing the MITM attack is $(S_1+S_2)n'+((S_1+S_2)\xi+S_1S_22^{-\xi}n')\cdot(\frac{2}{1+\delta'})^{\xi}$. We include here the cost of the pre-computation phase and the actual MITM cost. We obtain that the time complexity is $O((S_1+S_2)n'+(S_1+S_2)\xi(\frac{2}{1+\delta'})^{\xi}+S_1S_2n'(\frac{1}{1+\delta'})^{\xi})$. Taking again $n'=n-(a-1)2^b$, $k'=b$, $\delta'=\delta^{2^{a-1}}$, $\Delta=\delta$, we obtain the following result for MITM.

Theorem 10. *Let $n'=8\ln(\frac{2}{\theta}S_1S_2)\delta^{-2^a}$. Take k_1 and k_2 values such that $b=k_1+k_2$. Let $S_j=\sum_{i=0}^{\mathsf{max}_{\mathsf{HW}}(k_j)}\binom{k_j}{i}$ where $\mathsf{max}_{\mathsf{HW}}(k_j)=k_j(\frac{1-\Delta}{2})+\frac{\sigma'}{2}\sqrt{k_j}$ for $j\in\{1,2\}$. For $k\leq a\cdot b$ and a secret s that is Δ-sparse, the MITM algorithm heuristically $(n=8\ln(\frac{2}{\theta}S_1S_2)\delta^{-2^a}+(a-1)2^b, t=O(kan+(S_1+S_2)n'+(S_1+S_2)\xi(\frac{2}{1+\delta^{2^{a-1}}})^{\xi}+S_1S_2n'(\frac{1}{1+\delta^{2^{a-1}}})^{\xi}), m=kn+S_2+(S_1+S_2)n', \theta, b)$-solves the* LPN *problem.*

4.3 Gaussian Elimination

In the case of a sparse noise, one may try to recover the secret s by using Gaussian elimination. It is well known that LPN with noise 0, i.e. $\tau=0$, is an easy problem. If we are given $\Theta(k)$ queries for which the noise is 0, one can just run Gaussian elimination and in $O(k^3)$ recover the secret s. For a $\mathsf{LPN}_{k,\tau}$ instance, the event of having no noise for k queries happens with a probability $p_{\mathsf{nonoise}}=(1-\tau)^k$.

We design the following algorithm for solving LPN: first, we have no reduction phase. For each k new queries, we assume that the noise is 0. We recover an ν through Gaussian elimination. We must test if this value is the correct secret by computing the Hamming weight of $A'\nu+b'$, where A' is the matrix that contains n' fresh queries and b' is the vector containing the corresponding noisy inner products.

We expect to have a Hamming weight in the range $[0, (\frac{1-\delta}{2})n' + \sigma\frac{\sqrt{n'}}{2}]$, where σ is a constant. From the previous results we know that for a correct secret we have

$$\Pr[HW(A's+b') \text{ not in range}] \leq e^{-\frac{\sigma^2}{2}}.$$

If we want to bound by $\theta/2$ the probability that the Hamming weight of the noise is not in the correct range, for the correct secret, we obtain that $\sigma = \sqrt{2\ln(\frac{2}{\theta})}$.

For a $\nu \neq s$, we use the Hoeffding inequality to bound that $HW(A'\nu + b')$ is in the correct range. Let $X_1, \ldots, X_{n'}$ be the random variables that correspond to $X_i = \langle v_i, \nu \rangle \oplus b_i$. Let $X = X_1 + \ldots + X_{n'}$. We have $E(X) = \frac{n'}{2}$. Using the Hoeffding inequality, we take $t = \frac{\delta n'}{2} - \sigma\frac{\sqrt{n'}}{2}$ and obtain

$$\begin{aligned}\Pr[\text{failure}] &= 2^k \Pr[HW(A'\nu + b')] \text{ in correct range}]\\ &= 2^k \Pr[X - E(X) \leq -t]\\ &\leq 2^k e^{-\frac{2(\frac{\delta n'}{2} - \sigma\frac{\sqrt{n'}}{2})^2}{n'}} = 2^k e^{-\frac{(\delta\sqrt{n'}-\sigma)^2}{2}}\end{aligned}$$

If we bound this probability of failure by $\theta/2$ we obtain that we need at least $n' = (\sqrt{2\ln\frac{2^{k+1}}{\theta}} + \sigma)^2\delta^{-2}$ queries besides the k that are used for the Gaussian elimination.

As aforementioned, with a probability of $p_{\mathsf{nonoise}} = (1-\tau)^k$, the Gaussian elimination will give the correct secret. Thus, we have to repeat our algorithm $\frac{1}{p_{\mathsf{nonoise}}}$ times.

Complexity analysis. The computation of the Hamming weight has a cost of $O(n'k^2)$. Given that we run the Gaussian elimination and the verification step $\frac{1}{p_{\mathsf{nonoise}}}$ times, we obtain the following theorem for this algorithm:

Theorem 11. *Let $n' = \left(\sqrt{2\ln\frac{2^{k+1}}{\theta}} + \sqrt{2\ln(\frac{2}{\theta})}\right)^2 \delta^{-2}$ and let c be a constant. The Gaussian elimination algorithm ($n = \frac{k+c}{(1-\tau)^k} + n' + c, t = O\left(\frac{n'k^2+k^3}{(1-\tau)^k}\right), m = k^2 + n'k, \theta, k$)-solves the* LPN *problem.*[6]

Remark 2. Notice that this algorithm recovers the whole secret at once and the only assumption we make is that the noise is sparse. We don't need to run the transform such that we have a sparse secret and there are no queries lost during the reduction phase.

Remark 3. In the extreme case where $(1-\tau)^k > \theta$ the Gaussian elimination algorithm can just assume that k queries have noise 0 and retrieve the secret s without verifying that this is the correct secret.

5 Tightness of Our Query Complexity

In this section we compare the theoretical analysis with implementation results of all the LPN solving algorithms described in Sections 3 & 4.

We implemented the BKW, LF1 and LF2 algorithms as they are presented in [32] and in pseudocode in Algorithms 1-3. The implementation was done in C on a Intel Xeon 3.33Ghz CPU. We used a custom bit library to store and handle bit vectors. Using the OpenMP library[7], we have also parallelized certain crucial parts of the algorithms. The xor-ing in the reduction phases as well as the majority phases for instance, are easily distributed onto multiple threads to speed up the computation. Furthermore, we implemented the exhaustive search and MITM algorithms described in Section 4.

[6] Given that we receive uniformly distributed vectors from the LPN oracles, we expect to need $n+2$ vectors v to have n linearly independent ones. We express this by the use of the constant c.

[7] `http://openmp.org/wp`

The various matrix operations performed for the sparse LPN solving algorithms are done with the M4RI library [8]. Regarding the memory model used, we implemented the one described in [32] in order to accommodate the LF2 algorithm. The source code of our implementation can be found at `http://lasec.epfl.ch/lpn/lpn_source_code.zip`.

We ran all the algorithms for different LPN instances where the size of the secret varies from 32 to 100 bits and the Bernoulli parameter τ takes different values from 0.01 to 0.4. A value of $\tau = 0.1$ for a small k as the one we are able to test means that very few, if none, of the queries have the noise bits set on 1. For this sparse case, an exhaustive search is the optimal strategy. Also, $\tau = 0.4$ might seem also as an extreme case. Still, we provide the query complexity for these extreme cases to fully observe the behaviour of the LPN solving algorithms.

For each LPN instance, we try to find the theoretical number of oracle queries required to get a 50% probability of recovering the full secret while optimizing the time complexity. This means that in half of our instances we recover the secret correctly. In the other of the cases it may happen that one or more bits are guessed wrong. We thus take $\theta = \frac{1}{3}$ as the probability of failure for the first block. We choose a and b that would minimize the time complexity and we apply this split in our theoretical bounds in order to compute the theoretical number of initial queries. We apply the same split in practice and try to minimize the number of initial queries such that we maintain a 50% probability of success. We thus experimented with different values for the original number of oracle samples, and ran multiple instances of the algorithms to approximate the success probability. One can observe that in our practical and theoretical results the a,b parameters are the same and the comparison is consistent. We were limited by the power of our experimental environment and thus we were not able to provide results for instances that require more than 2^{30} queries.

5.1 BKW*

The implementation results for BKW* are presented in Table 3. Each entry in the table is of the form $\log_2(n)(a)$, where n is the number of oracle queries that were required to obtain a 50% success rate for the full recovery of the secret. Parameter a is the algorithm parameter denoting the number of blocks into which the vectors were split. We take $b = \lceil \frac{k}{a} \rceil$. By maintaining the value of a, we can easily compute the number of queries and the time & memory complexity. In Table 4 we present the theoretical results for BKW* obtained by using Theorem 4. We can see that our theoretical and practical results are within a factor of at most 2.

τ	k				
	32	48	64	80	100
0.01	10.40(5)	11.85(6)	15.01(6)	17.68(7)	20.78(7)
0.10	14.32(4)	19.99(4)	23.13(4)	27.30(4)	
0.20	18.64(3)	23.84(3)			
0.25	21.93(2)	25.95(3)			
0.40	27.25(2)				

Table 3: BKW* query complexity - practice

τ	k				
	32	48	64	80	100
0.01	10.97(5)	12.82(6)	15.93(6)	18.66(7)	21.74(7)
0.10	15.84(4)	20.01(4)	24.12(4)	28.20(4)	33.28(4)
0.20	19.71(3)	24.85(3)	30.97(3)	34.83(4)	39.90(4)
0.25	21.81(2)	26.95(3)	33.07(3)	38.14(3)	44.11(4)
0.40	28.24(2)	36.38(2)	43.64(3)	48.71(3)	55.78(3)

Table 4: BKW* query complexity - theory

[8] `http://m4ri.sagemath.org/`

If we take the example of $\mathsf{LPN}_{100,0.01}$, we need $2^{20.78}$ queries and our theoretical analysis gives a value of $2^{21.47}$. These two values are very close compared with the value predicted by [32], $2^{25.64}$, which is a factor 10 larger. We emphasize again that for both the theory and the practice we use the split that optimizes the time complexity and from this optimal split we derive the number of queries.

Remark 4. For the BKW^* algorithm we tried to optimize the average final bias of the queries, i.e. obtaining a better value than $\delta^{2^{a-1}}$. Recall that at the beginning of the reduction phase, we order the queries in equivalence classes and then choose a representative vector that is xored with the rest of queries from the same class. One variation of this reduction operation would be to change several times the representative vector. The incentive for doing so is the following: one representative vector that has error vector set on 1 affects the bias δ of all queries, while by choosing several representative vectors this situation may be improved; more than half of them will have error bit on 0. We implemented this new approach but we found that it does not bring any significant improvement. Another change that was tested was about the majority rule applied during the solving phase. Queries have a worst case bias of $\delta^{2^{a-1}}$ (See Lemma 2), but some have a larger bias. So, we could apply a weighted majority rule. This would decrease the number of queries needed for the solving phase. Unfortunately we implemented the idea and discovered that the complexity advantage is very small.

5.2 LF1

Below we present the experimental and theoretical results for the LF1 algorithm. As a first observation we can see that, for all instances, this algorithm is a clear optimization over the original BKW^* algorithm. As before, each entry is of the form $\log_2(n)(a)$, where n and a are selected to obtain a 50% success rate for the full recovery of the secret and $b = \lceil \frac{k}{a} \rceil$.

τ	k				
	32	48	64	80	100
0.01	7.32(6)	10.12(6)	11.58(7)	13.32(8)	14.99(8)
0.10	10.20(4)	13.20(4)	15.52(5)	17.98(5)	21.38(5)
0.20	11.53(3)	15.57(3)	18.03(4)	21.04(4)	25.18(4)
0.25	12.69(3)	16.20(3)	20.70(4)	22.24(4)	25.93(4)
0.40	15.61(2)	19.74(2)	23.97(3)		

Table 5: LF1 query complexity - practice

τ	k				
	32	48	64	80	100
0.01	8.89(6)	10.53(6)	12.77(7)	14.17(8)	16.13(8)
0.10	11.38(4)	13.87(4)	17.04(5)	18.56(5)	22.05(5)
0.20	13.01(3)	17.06(3)	19.05(4)	21.77(4)	26.59(4)
0.25	14.42(3)	17.25(3)	22.65(4)	23.39(4)	26.72(4)
0.40	16.95(2)	24.01(2)	25.83(3)	28.30(3)	35.00(3)

Table 6: LF1 query complexity - theory

Table 6 shows our theoretical results for LF1 using Theorem 5. When we compare the experimental and the practical results for LF1 (See Table 5 and Table 6) we can see that the gap between them is of a factor up to 3.

Remark 5. One may observe a larger difference for the $\mathsf{LPN}_{48,0.4}$ instance: $n = 2^{19.74}$ (practice) vs. $n = 2^{24.01}$ (theory). For this case, the implementation requires $n = 2^{19.74}$ initial queries compared with the theory that requires $n = 2^{24.01}$ queries. Here we have $a = 2$ and $b = 24$ and the term $(a-1)2^b$ dominates the query complexity. The discrepancy comes from the worst-case analysis of the reduction phase where we say that at each reduction step we discard 2^b queries. With this reasoning, we predict to lose 2^{24} queries. If we analyse more closely, we discover that actually in the average-case we discard only $2^b \cdot \left[1 - \left(1 - \frac{1}{2^b}\right)^n\right]$ queries (this is the number of expected non-empty equivalence classes). Thus, with only

$2^{19.74}$ initial queries, we run the reduction phase and discard $2^{19.70}$ queries, instead of 2^{24}. We are left with $2^{14.45}$, queries which are sufficient for the solving phase. We note that for large LPN instances, this difference between worst-case and average-case analysis for the number of deleted queries during reduction rounds becomes negligible.

Remark 6. Recall that in LF1, like in all LPN solving algorithms, we perform the reduction phase by splitting the queries into a blocks of size b. When this split is not possible, we consider that we have $a-1$ blocks of size b and a last block shorter of size b' with $b' < b$. By LF1* we denote the same LPN solving algorithm that makes use of the Walsh transform but where the split of the blocks is done different. We allow now to have a last block larger than the rest. The gain for this strategy may be the following: given that we recover a larger block of the key, we run our solving phase fewer times. Although the complexity of the transform is bigger as we work with a bigger block, the reduction phase has to be applied fewer times. From our experiments we discover there seems to be no difference between the performance of the two algorithms.

5.3 LF2

We tested the LF2 heuristic on the same instances as for BKW* and LF1. The results are summarized in Table 7. To illustrate the performance of the heuristic, we concentrate on a particular instance, $\mathsf{LPN}_{100,0.1}$ with $a = 5, b = 20$. As derived in [32], the LF1 algorithm for this parameter set should require less than $(8 \cdot b + 200) \cdot \delta^{-2^a} \approx 2^{18.77}$ queries for a solving phase and $(a-1) \cdot 2^b + (8 \cdot b + 200) \cdot \delta^{-2^a} \approx 2^{22.13}$ queries overall to achieve a success probability of 50%. Using our theoretical analysis, the LF1 algorithm for this parameter set requires to have $8 \ln(3 \cdot 2^b)\delta^{-2^a} + (a-1)2^b \approx 2^{22.05}$ queries overall and $2^{17.20}$ queries for the solving phase. Our experimental results for LF1 were a bit lower than our theoretical ones: $2^{21.38}$ oracle samples were sufficient. If we use the LF2 heuristic starting with $3 \cdot 2^{20} \approx 2^{21.58}$ samples, we get about the same amount of vectors for the solving phase. In this case there are no queries lost during reduction. We thus have much more queries than should actually be required for a successful solving phase and correctly solve the problem with success probability close to 100%. So we can try to start with less. By starting off with $2^{20.65}$ queries and thus loosing some queries in each reduction round, we also solved the LPN problem in slightly over 50% of the cases. The gain in total query complexity for LF2 is thus noticeable but not extremely important.

As another example, consider the parameter set $k = 768, \tau = 0.05$ proposed at the end of [32]. The values for a, b which minimize the query complexity are $a = 9, b = 86$ ($a \cdot b = 774 > k$). Solving the problem with LF1 should thus require about 2^{87} vectors for the solving phase and 2^{89} oracle samples overall. Using LF2, as $3 \cdot 2^b \approx 2^{87}$ oracle samples would be sufficient, we obtain a reduction by a factor ≈ 4.

Even though LF2 introduces linear dependencies between queries, this doesn't seem to have any noticeable impact on the success probability in recovering the secret value.

Remark 7. A general observation for all these three algorithms, shown also by our results, is that the bias has a big impact on the number of queries and the complexity. Recall that the bias has value $\delta^{2^{a-1}}$ at the end of the reduction phase. We can see from our tables that the lower the value of τ, i.e. larger value of $\delta = 1 - 2\tau$, the higher a can be chosen to solve the LPN instance. Also, for a constant τ, the higher the size of the secret, i.e. the lower the noise, the higher a can be chosen.

Remark 8. The LF2 algorithm is a variation of LF1 that offers a different heuristic technique to decrease the number of initial queries. The same trick could be used for BKW*, exhaustive search and MITM.

While the same analysis can be applied for exhaustive search and MITM as for LF2, BKW* is a special case. We denote by BKW^2 this variation of BKW where we use the reduction phase from LF2. Recall that for BKW* we need to have $n = 2^{b+1}\delta^{-2^a} \ln(\frac{b}{\theta}) + (a-1)2^b$ queries and here the dominant

τ	k				
	32	48	64	80	100
0.01	6.85(6)	9.09(6)	10.24(7)	12.41(8)	13.15(8)
0.10	9.30(4)	12.60(4)	15.12(5)	16.90(5)	20.65(5)
0.20	10.88(3)	15.40(3)	16.94(4)	20.47(4)	24.88(4)
0.25	12.34(3)	15.92(3)	20.61(4)	21.00(4)	25.40(4)
0.40	15.44(2)	19.74(2)	23.52(3)		

Table 7: LF2 query complexity - practice

term is $2^{b+1}\delta^{-2^a}\ln(\frac{b}{\theta})$. Thus, we need to start with $3 \cdot 2^b + \varepsilon$, where $\varepsilon > 0$ and increase such that at the end of the last iteration of the reduction we get the required number of queries. This improves the initial number of queries and we have a gain of factor a for the time complexity. For an $\mathsf{LPN}_{48,0.1}$ instance, our implementation of BKW^2 requires $n = 2^{13.82} = 3.54 \cdot 2^{12}$ initial queries and increases it, during the reduction phase, up to $2^{19.51}$, the amount of queries needed for the solving phase. Thus, there is an improvement from $2^{19.99}$ (See Table 3) to $2^{13.82}$ and the time complexity is better. While this is an improvement over BKW^*, it still performs worse than LF1 and LF2.

5.4 Exhaustive search

Recall that for exhaustive search we have two variants. The results for Search_1 are displayed in Table 8 and Table 9. For Search_1 we observe that the gap between theory and practice is of a factor smaller than 4. In terms of number of queries, Search_1 brings a small improvement compared to LF1. We will see in the next section the complete comparison between all the implemented algorithms. The same $(a-1)2^b$ dominant term causes the bigger difference for the instances $\mathsf{LPN}_{48,0.4}$ and $\mathsf{LPN}_{64,0.25}$.

τ	k				
	32	48	64	80	100
0.01	5.16(1)	5.70(1)	6.12(1)	13.25(8)	14.93(8)
0.10	10.15(4)	13.15(4)	16.44(4)	17.93(5)	21.34(5)
0.20	11.51(3)	15.54(3)	17.99(4)	21.02(4)	25.15(4)
0.25	12.66(3)	16.18(3)	19.88(3)		
0.40	15.61(2)	19.74(2)			

Table 8: Search_1 query complexity - practice

τ	k				
	32	48	64	80	100
0.01	5.16(1)	5.70(1)	6.12(1)	14.05(8)	16.06(8)
0.10	11.33(4)	13.84(4)	17.61(4)	18.50(5)	22.04(5)
0.20	13.01(3)	17.06(3)	18.99(4)	21.76(4)	26.59(4)
0.25	14.42(3)	17.25(3)	23.01(3)	28.00(3)	26.71(4)
0.40	16.98(2)	24.01(2)	25.87(3)	28.31(3)	35.00(3)

Table 9: Search_1 query complexity - theory

The results for Search_2 are displayed in Table 10 and Table 11.

We notice that for both Search_1 and Search_2 the instances $\mathsf{LPN}_{32,0.01}$, $\mathsf{LPN}_{48,0.01}$ and $\mathsf{LPN}_{68,0.01}$ have the number of queries very low. This is due to the following observation: for $n \leq 68$ linearly independent queries and $\tau = 0.01$ we have that the noise bits are all 0 with a probability larger than 50%. Thus, for $k \leq 64$ we hope that the $k + c$ queries we receive from the oracle have all the noise set on 0, where c is a constant. With k noiseless, linearly independent queries we can just recover s with Gaussian elimination. This is an application of Remark 3.

τ	k				
	32	48	64	80	100
0.01	5.16(1)	5.70(1)	6.12(1)	13.25(8)	14.93(8)
0.10	10.15(4)	13.15(4)	15.36(5)	17.93(5)	21.34(5)
0.20	11.51(3)	15.54(3)	17.99(4)	21.02(4)	25.15(4)
0.25	12.66(3)	16.18(3)	20.63(4)		
0.40	15.61(2)	19.74(2)			

Table 10: Search_2 query complexity - practice

τ	k				
	32	48	64	80	100
0.01	5.16(1)	5.70(1)	6.12(1)	14.05(8)	16.06(8)
0.10	11.33(4)	13.84(4)	16.89(5)	18.50(5)	22.04(5)
0.20	13.01(3)	17.06(3)	18.99(4)	21.76(4)	26.59(4)
0.25	14.42(3)	17.25(3)	22.63(4)	23.38(4)	26.71(4)
0.40	16.98(2)	24.01(2)	25.87(3)	28.31(3)	35.00(3)

Table 11: Search_2 query complexity - theory

5.5 MITM

In the case of MITM, the experimental and theoretical results are illustrated in Table 12 and Table 13. There is a very small difference between MITM and exhaustive search algorithms for a sparse secret: in practice, MITM requires just couple of tens queries less than Search_1 and Search_2 for the same *a* and *b* parameters.

τ	k				
	32	48	64	80	100
0.01	5.16(1)	5.70(1)	6.12(1)	13.25(8)	14.93(8)
0.10	10.13(4)	13.15(4)	16.47(4)		
0.20	11.49(3)	15.54(3)			
0.25	12.89(2)				
0.40					

Table 12: MITM query complexity - practice

τ	k				
	32	48	64	80	100
0.01	5.16(1)	5.70(1)	6.12(1)	14.10(8)	16.10(8)
0.10	11.37(4)	13.87(4)	17.61(4)	21.59(4)	22.05(5)
0.20	13.02(3)	17.06(3)	23.00(3)	28.00(3)	26.59(4)
0.25	16.03(2)	17.26(3)	23.01(3)	28.00(3)	35.00(3)
0.40	16.98(2)	24.01(2)	25.87(3)	28.31(3)	35.00(3)

Table 13: MITM query complexity - theory

5.6 Gaussian Elimination

As aforementioned, in the Gaussian elimination the only assumption we need is to have a noise sparse. We don't run any reduction technique and the noise is not affected. As the algorithm depends on the probability to have a 0 noise on k linearly independent vectors, the complexity decays very quickly once we are outside the sparse noise scenario. We present below the theoretical results obtained for this algorithm.

In the next section we will show the effectiveness of this simple idea in the sparse case scenario and compare it to the other LPN solving algorithms.

Again for $\mathsf{LPN}_{32,0.01}$, $\mathsf{LPN}_{48,0.01}$ and $\mathsf{LPN}_{64,0.01}$ we apply Remark 3.

5.7 Covering Codes

The covering code requires the existence of a code with the optimal coverage. For each instance one has to find an optimal code that minimizes the query and time complexity. Unlike the previous algorithms,

τ	k				
	32	48	64	80	100
0.01	5.16	5.70	6.12	8.43	8.89
0.10	10.04	12.91	15.73	18.48	21.84
0.20	15.31	21.04	26.60	32.08	38.84
0.25	18.28	25.51	32.56	39.52	48.15
0.40	28.58	40.96	53.17	65.28	80.34

Table 14: Gaussian elimination query complexity - theory

this algorithm cannot be truly automatized. In practice we could test only the cases that were suggested in [23]. Thus, we are not able to compare the theoretical and practical values. Nevertheless, we will give theoretical values for different practical parameters in the next section.

6 Complexity Analysis of the LPN Solving Algorithms

We have compared our theoretical bounds with our practical results and we saw that there is a small difference between the two. Our theoretical analysis also gives tighter bounds compared with the results from [32]. We now extend our theoretical results and compare the asymptotic performance of all the LPN algorithms for practical parameters used by the LPN-based constructions. We consider the family of $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$ instances proposed in [3,14]. Although the covering code cannot be automatized, as for each instance we have to try different codes with different sizes and dimensions, we provide results also for this algorithm. When dealing with the covering code reduction, we always assume the existence of an ideal code and compute the bias introduced by this step. We do not consider here concatenation of ideal codes and we will see that we obtain a worse result for the $\mathsf{LPN}_{512,0.125}$ instance, although the difference is small. We also stick with the BKW reduction steps and don't use the LF2 reduction. As aforementioned, the LF2 reduction brings a small improvement to the final complexity. This does not affect the comparison between all the LPN solving algorithms.

We analyse the time complexity of each algorithm, by which we mean the number of bit operations the algorithm performs while solving an LPN problem. For each algorithm, we consider values of k for which the parameters (a,b) minimising the time complexity are such that $k = a \cdot b$. For the LF2 algorithm, we select the initial number of queries such that we are left with at least $n' = 8\ln(3 \cdot 2^b)\delta^{-2^a}$ queries after the reduction phase. Recall that by Search_1 we denote the standard exhaustive search algorithm and Search_2 is making use of a Walsh-Hadamard transform. The results are illustrated in Figure 1. We recall the time complexity and the initial number of queries for each algorithm in Table 15, where S represents the number of sparse secrets with $S < 2^b$. For MITM, the values S_1 (resp. S_2) represent the number of possible values for the first (resp. second) half of the secret, $n' = 8(\ln(6S_1S_2))\delta^{-2^a}$ represents the number of queries left after the reduction phase and ξ represents the Hamming weight of the mask used. Recall that θ is $\frac{1}{3}$.

We can bound the logarithmic complexity of all these algorithms by $\frac{k}{\log_2(k)} + c_1$ and $\log_2(k) + \sqrt{k} + c_2$. The lower bound is given by the asymptotic complexity of the Gaussian elimination that can be expressed as $\log_2 k + \sqrt{k}$ when $\tau = \frac{1}{\sqrt{k}}$.

The complexity of BKW can be written as $\min_{k=ab}(\mathsf{poly} \cdot 2^b \cdot \delta^{-2^a})$ and for the other algorithms the formula is $\min_{k=ab}(\mathsf{poly} \cdot (2^b + \delta^{-2^a}))$, where poly denotes a polynomial factor. By searching for the optimal a,b values, for $a > 1$, we find $a \sim \log_2 \frac{k}{(\log_2 k)^2 \ln \frac{1}{\delta}}$ and $b = \frac{k}{a}$ and obtain that 2^b dominates δ^{-2^a}. For $\delta = 1 - \frac{2}{\sqrt{k}}$ we obtain the complexity $\mathsf{poly} \cdot 2^{\frac{k}{\log_2(k)}}$. For the case where $a = 1$, we have that the complexity

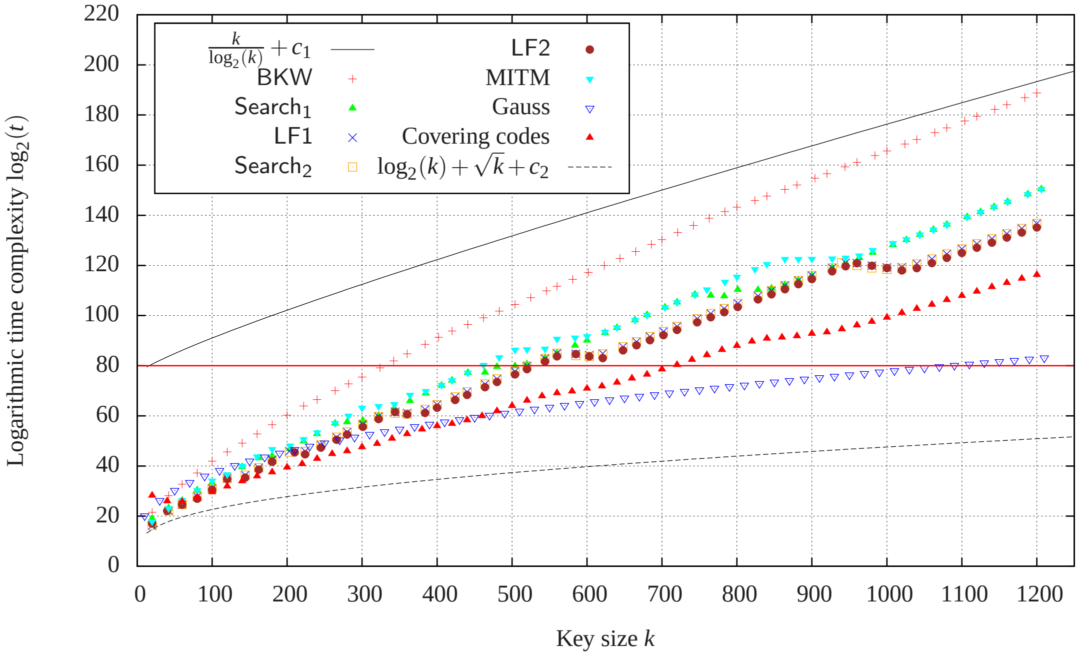

Fig. 1: Time Complexity of LPN Algorithms on instances $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$

of BKW is $\mathrm{poly}\cdot 2^k$, while for LF1, LF2, Search$_2$ we have $\mathrm{poly}+k2^k$. A more special analysis needs to be done for the Search$_1$ and $MITM$: here we have that the complexity is $\mathrm{poly}\cdot S_r$ and $\mathrm{poly}\cdot S^2_{r'}$, respectively, where we define S_r to be $\#\{v\in\{0,1\}^k \mid HW(v)\leq r\}$. We need to bound the value of S_r. By induction we can show that $S_r \leq \frac{k}{k-r-1}\cdot\frac{k^r}{r!}$. For $\tau\approx\frac{1}{\sqrt{k}}$, we have that $r\approx(1+\frac{\sigma}{2})\sqrt{k}$ and $r'\approx(\frac{1}{2}+\frac{\sigma}{2\sqrt{2}})\sqrt{k}$. We obtain that the complexity for both algorithms is $\mathrm{poly}\cdot 2^{\gamma\sqrt{k}\log_2 k+O(\sqrt{k})}$, where γ is a constant. This is not better than $2^{\frac{k}{\log_2(k)}}$ for $k<200000$, but asymptotically this gives a better complexity.

We see that in some cases increasing the value of k may result in a decrease in time complexity. The reason for this is that we are considering LPN instances where the noise parameter τ takes value $\frac{1}{\sqrt{k}}$. Thus, as k grows, the noise is reduced, which leads to an interesting trade-off between the complexity of the solving phase and the complexity of the reduction phase of the various algorithms. This behaviour does not seem to occur for the BKW algorithm. In this case, the query complexity $n=2^{b+1}(1-\frac{2}{\sqrt{k}})^{-2^a}\ln(2k)+(a-1)2^b$ is largely dominated by the first term, which grows exponentially not only in terms of the noise parameter, but also in terms of the block size b.

*Remark 9 (*LF1 *vs.* Search$_2$*).* As shown in Figure 1, the overall complexity of the LF1 and Search$_2$ algorithms is quasi identical. From Theorems 5 and 9, we deduce that for the same parameters (a,b), the Search$_2$ algorithm should perform better as long as $S<2^{b-1}$. This is indeed the case for the instances we consider here, although the difference in complexity is extremely small.

We can see clearly that for the $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$ family of instances, the Gaussian elimination outperforms all the other algorithms for $k>500$. For any $k<1000$, the $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$ does not offer an 80 bit security. This requirement is achieved for $k=1090$.

Selecting secure parameters. We remind that for each algorithm we considered, our analysis made use of a heuristic assumption of query and noise independence after reduction. In order to propose security parameters, we simply consider the algorithm which performs best under this assumption.

LPN algorithm	Query complexity(n)	Time complexity(t)
BKW	$2^{b+1}\delta^{-2^a}\ln(\frac{b}{\theta})+(a-1)2^b$	kan
LF1	$8\ln(\frac{2^b}{\theta})\delta^{-2^a}+(a-1)2^b$	$kan+b2^b$
Search$_1$	$8\ln(\frac{2S}{\theta})\delta^{-2^a}+(a-1)2^b$	$kan+8\ln(\frac{2S}{\theta})\delta^{-2^a}S$
LF2	$3\cdot 2^b \geq 8\ln(\frac{2^b}{\theta})\delta^{-2^a}$	$kan+b2^b$
Search$_2$	$8\ln(\frac{2S}{\theta})\delta^{-2^a}+(a-1)2^b$	$kan+b2^b$
MITM	$8\ln(\frac{2}{\theta}S_1S_2)\delta^{-2^a}+(a-1)2^b$	$kan+\mathsf{comp}+\mathsf{comp}_{\mathsf{mitm}}$ where $\mathsf{comp}=(S_1+S_2)n'$ and $\mathsf{comp}_{\mathsf{mitm}}=(S_1+S_2)\xi(\frac{2}{1+\delta^{2^{a-1}}})^{\xi}+S_1S_2n'(\frac{1}{1+\delta^{2^{a-1}}})^{\xi}$
Gaussian elimination	$\frac{k}{(1-\tau)^k}+(\sqrt{2\ln\frac{2^{k+1}}{\theta}}+\sigma)^2\delta^{-2}$ where $\sigma=\sqrt{2\ln(\frac{2}{\theta})}$	$\frac{(\sqrt{2\ln\frac{2^{k+1}}{\theta}}+\sigma)^2\delta^{-2}k^2+k^3}{(1-\tau)^k}$

Table 15: Query & Time complexity for LPN solving algorithms for recovering the first b bits

By taking all the eight algorithms described in this article, Tables 16-23 display the logarithmic time complexity for various LPN parameters. For instance, the LF2 algorithm requires 2^{84} steps to solve a $\mathsf{LPN}_{384,0.25}$ instance.

We recall here the result from [23]: an instance $\mathsf{LPN}_{512,0.125}$ offers a security of 79.7. We obtain a value of 82. The difference comes mainly from the use of LF2 reduction in [23] and from a search of optimal concatenation of linear codes.

When comparing all the algorithms, we have to keep in mind that the Gaussian elimination recovers the whole secret, while for the rest of the algorithms we give the complexity to recover a block of the secret. Still, this does not affect our comparison as we have proven in Section 3 that the complexity of recovering the first block dominates the total complexity.

We highlight with red the best values obtained for different LPN instances. We observe the following behaviour: for a sparse case scenario, i.e. $\tau = 0.05$ or $\tau = \frac{1}{\sqrt{k}} < 0.05$, the Gaussian elimination offers the best performance and no k from our tables offers a 80 bit security. Once we are outside the sparse case scenario, we have that LF2 and the covering code algorithms are the best ones. The covering code proves to be better than LF2 for a level of noise of 0.125. While the performance of the covering code reduction highly depends on the sparseness of the noise, LF2 has a more general reduction phase and is more efficient for noise parameters of 0.25 and 0.4. Also for a $\tau > 0.5$ the covering code is better than the Gaussian elimination.

Thus, for different scenarios, there are different algorithms that prove to be efficient. This comparison clearly shows that for the family of instances $\mathsf{LPN}_{k,\frac{1}{\sqrt{k}}}$ neither the BKW, nor its variants are the best ones. One should use the Gaussian elimination algorithm.

As we have shown, there still remains a small gap between the theoretical and practical results for the algorithms we analysed. It thus seems reasonable to take a safety margin when selecting parameters to achieve a certain level of security.

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	69	88	97	106	114	123	140	198
0.05	67	88	98	109	118	127	145	216
0.125	79	105	116	128	138	149	170	253
0.25	93	123	137	150	163	175	201	295
0.4	115	147	163	180	196	212	244	347

Table 16: Security of LPN against the BKW algorithm

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	50	63	71	79	85	88	102	145
0.05	50	62	71	79	87	95	102	159
0.125	56	73	78	88	98	107	125	176
0.25	64	84	89	100	110	121	142	199
0.4	76	94	103	116	129	142	168	229

Table 17: Security of LPN against the LF1 algorithm

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	49	61	69	78	85	86	100	143
0.05	49	61	69	78	86	94	100	158
0.125	55	73	77	87	97	106	124	175
0.25	64	84	88	99	109	121	142	198
0.4	76	94	103	116	129	141	168	229

Table 18: Security of LPN against the LF2 algorithm

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	56	69	77	80	87	95	108	154
0.05	51	69	78	84	89	95	111	162
0.125	64	82	91	100	110	121	140	199
0.25	82	110	122	134	145	155	179	263
0.4	109	141	157	173	189	205	236	337

Table 19: Security of LPN against the Search_1 algorithm

Based on this analysis, we could recommend the LPN instances $\mathsf{LPN}_{512,0.25}$, $\mathsf{LPN}_{640,0.125}$, $\mathsf{LPN}_{1200,0.05}$ or $\mathsf{LPN}_{1280,\frac{1}{\sqrt{1280}}}$ to achieve 80 bit security for different noise levels. We note that the value $\mathsf{LPN}_{768,0.05}$ that Levieil and Fouque suggest as a secure instance to use actually offers only 66 bit security and thus is not recommended.

7 Conclusion

In this article we have analysed and presented the existing LPN algorithms in a unified framework. We introduced a new theoretical analysis and this has improved the bounds of Levieil and Fouque [32]. In order to give a complete analysis for the LPN solving algorithms, we also presented three algorithms that use the advantage that the secret is sparse. We analysed also the latest algorithm presented at Asiacrypt'14. While the covering code and the LF2 algorithms perform best in the general case where the Bernoulli noise parameter is constant, the Gaussian elimination shows that for the sparse case scenario the length of the secret should be bigger than 1100 bits. Also, we show that some values proposed by Leviel and Fouque are insecure in the sparse case scenario.

Acknowledgements We would like to thank Thomas Johansson and all the authors of [23] for their help in providing us with their paper and for their useful discussions. We further congratulate them for receiving the Best Paper Award of Asiacrypt'14.

References

1. Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Lazy Modulus Switching for the BKW Algorithm on LWE. In *Public Key Cryptography*, pages 429–445, 2014.

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	50	63	71	79	84	88	102	145
0.05	50	62	71	79	87	95	102	159
0.125	56	73	78	88	98	107	125	176
0.25	64	84	89	100	110	121	142	199
0.4	76	94	103	116	129	142	168	229

Table 20: Security of LPN against the Search_2 algorithm

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	56	70	78	86	91	96	111	159
0.05	55	70	78	88	98	104	114	176
0.125	65	88	96	104	112	122	142	203
0.25	85	113	125	137	148	159	184	270
0.4	109	141	158	174	190	206	237	339

Table 21: Security of LPN against the $\mathcal{MITM}$ algorithm

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	49	56	59	62	64	67	70	85
0.05	27	37	42	47	52	57	66	127
0.125	57	83	95	108	120	133	158	279
0.25	114	168	195	221	248	275	328	565
0.4	197	292	339	386	434	481	576	979

Table 22: Security of LPN against the Gaussian elimination algorithm

τ	k							
	256	384	448	512	576	640	768	1280
$\frac{1}{\sqrt{k}}$	44	55	59	64	70	73	85	123
0.05	42	54	59	65	72	78	88	132
0.125	52	67	74	82	89	96	109	161
0.25	70	87	96	106	115	125	139	204
0.4	94	110	123	136	149	161	179	281

Table 23: Security of LPN against the covering codes algorithm

2. MartinR. Albrecht, Carlos Cid, Jean-Charles Faugre, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. *Designs, Codes and Cryptography*, pages 1–30, 2013.
3. Michael Alekhnovich. More on Average Case vs Approximation Complexity. In *FOCS*, pages 298–307, 2003.
4. Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.
5. Sanjeev Arora and Rong Ge. New Algorithms for Learning in Presence of Errors. In *ICALP (1)*, pages 403–415, 2011.
6. Daniel J. Bernstein and Tanja Lange. Never trust a bunny. In *Radio Frequency Identification. Security and Privacy Issues - 8th International Workshop, RFIDSec 2012, Nijmegen, The Netherlands, July 2-3, 2012, Revised Selected Papers*, pages 137–148, 2012.
7. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In *CRYPTO*, pages 743–760, 2011.
8. Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic Primitives Based on Hard Learning Problems. In *CRYPTO*, pages 278–291, 1993.
9. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *STOC*, pages 435–440, 2000.
10. Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.
11. Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. HB^{++}: a Lightweight Authentication Protocol Secure against Some Attacks. In *SecPerU*, pages 28–33, 2006.
12. H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observables, 1952.
13. James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):pp. 297–301, 1965.
14. Ivan Damgård and Sunoo Park. Is Public-Key Encryption Based on LPN Practical? *IACR Cryptology ePrint Archive*, 2012:699, 2012.
15. Nico Döttling, Jörn Müller-Quade, and Anderson C. A. Nascimento. IND-CCA Secure Cryptography Based on a Variant of the LPN Problem. In *ASIACRYPT*, pages 485–503, 2012.
16. Alexandre Duc, Florian Tramèr, and Serge Vaudenay. Better Algorithms for LWE and LWR. In *To appear in EUROCRYPT*, 2015.

17. Alexandre Duc and Serge Vaudenay. HELEN: A Public-Key Cryptosystem Based on the LPN and the Decisional Minimal Distance Problems. In *AFRICACRYPT*, pages 107–126, 2013.
18. Robert Fitzpatrick. *Some Algorithms for Learning with Errors*. PhD thesis, Royal Holloway, University of London.
19. Marc P. C. Fossorier, Miodrag J. Mihaljevic, Hideki Imai, Yang Cui, and Kanta Matsuura. An Algorithm for Solving the LPN Problem and Its Application to Security Evaluation of the HB Protocols for RFID Authentication. In *INDOCRYPT*, pages 48–62, 2006.
20. Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. $\mathrm{HB}^{\#}$: Increasing the Security and Efficiency of HB^{+}. In *EUROCRYPT*, pages 361–378, 2008.
21. Elena Grigorescu, Lev Reyzin, and Santosh Vempala. On Noise-Tolerant Learning of Sparse Parities and Related Problems. In *Algorithmic Learning Theory - 22nd International Conference, 2011, Espoo, Finland, October 5-7, 2011. Proceedings*, pages 413–424, 2011.
22. Qian Guo, Thomas Johansson, and Carl Löndahl. A new algorithm for solving ring-lpn with a reducible polynomial. *CoRR*, abs/1409.0472, 2014.
23. Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, pages 1–20, 2014.
24. Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: An Efficient Authentication Protocol Based on Ring-LPN. In *FSE*, pages 346–365, 2012.
25. Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.
26. Nicholas J. Hopper and Manuel Blum. Secure Human Identification Protocols. In *ASIACRYPT*, pages 52–66, 2001.
27. Ari Juels and Stephen A. Weis. Authenticating Pervasive Devices with Human Protocols. In *CRYPTO*, pages 293–308, 2005.
28. Jonathan Katz, Ji Sun Shin, and Adam Smith. Parallel and Concurrent Security of the HB and HB^{+} Protocols. *J. Cryptology*, 23(3):402–421, 2010.
29. Eike Kiltz, Daniel Masny, and Krzysztof Pietrzak. Simple Chosen-Ciphertext Security from Low-Noise LPN. In *Public Key Cryptography*, pages 1–18, 2014.
30. Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient Authentication from Hard Learning Problems. In *EUROCRYPT*, pages 7–26, 2011.
31. Paul Kirchner. Improved Generalized Birthday Attack. *IACR Cryptology ePrint Archive*, 2011:377, 2011.
32. Éric Levieil and Pierre-Alain Fouque. An Improved LPN Algorithm. In *SCN*, pages 348–359, 2006.
33. Vadim Lyubashevsky. The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem. In *APPROX-RANDOM*, pages 378–389, 2005.
34. Vadim Lyubashevsky and Daniel Masny. Man-in-the-Middle Secure Authentication Schemes from LPN and Weak PRFs. In *CRYPTO (2)*, pages 308–325, 2013.
35. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding Random Linear Codes in $\tilde{O}(2^{0.054n})$. In *ASIACRYPT*, pages 107–124, 2011.
36. Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342. ACM, 2009.
37. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
38. Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications*, pages 106–113, 1988.
39. Gregory Valiant. Finding Correlations in Subquadratic Time, with Applications to Learning Parities and Juntas. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 11–20, 2012.

A Hoeffding's Bounds

Theorem 12. *[25] Let $X_1, X_2, \ldots, X_n$ be n independent variables. We are given that $\Pr[X_i \in [\alpha_i, \beta_i]] = 1$ for $1 \le i \le n$. We define $X = X_1 + \ldots + X_n$ and $E[X]$ is the expected value of X. We have that*

$$\Pr[X - E[X] \ge t] \le e^{-\frac{2t^2}{\sum_{i=1}^{n}(\beta_i - \alpha_i)^2}}$$

and

$$\Pr[X - E[X] \le -t] \le e^{-\frac{2t^2}{\sum_{i=1}^{n}(\beta_i - \alpha_i)^2}},$$

for any $t > 0$.

B LF1 - full recovery of the secret

We provide here an example of the LF1 algorithm, for the $\mathsf{LPN}_{512,0.125}$ instance, where we recover the full secret. We provide the values of a, b, n and time complexity to show that indeed the number of queries for the first iteration, dominates the number of queries needed later on. Also, this shows that the time complexity of recovering the first block dominates the total time complexity. For $\mathsf{LPN}_{512,0.125}$, we obtain the following values:

i	a	b	$\log_2 n$	$\log_2 t$
1	7	74	76.59	88.43
2	7	63	65.68	77.29
3	7	54	61.52	72.91
4	6	54	56.32	67.28
5	6	45	47.32	58.02
6	6	37	39.37	49.80
7	6	31	34.98	45.14
8	5	31	33.00	42.66
9	5	25	27.02	36.36
10	5	20	22.56	31.56
11	5	16	21.01	29.67
12	4	16	17.72	25.79
13	4	12	14.89	22.51
14	3	12	13.30	20.19
15	2	11	11.38	17.36
16	2	6	9.26	14.10
17	1	5	8.30	11.69

Table 24: Full secret recovery for the instance $\mathsf{LPN}_{512,0.125}$

The way one can interpret this table is the following: LF1 recovers first 74 bits by taking $a = 7$ and requiring $2^{76.59}$ queries. The total complexity of this step, i.e. the reduction, solving and updating operation, is of $2^{88.43}$ bit operations. Next, LF1 solves $\mathsf{LPN}_{438,0.125}$ and continues this process until it recovers the whole secret.

We can easily see that indeed the number of queries and the time complexity of the first block dominate the other values.