Optimization of LPN Solving Algorithms

Sonia Bogos and Serge Vaudenay

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

Abstract. In this article we focus on constructing an algorithm that automatizes the generation of LPN solving algorithms from the considered parameters. When searching for an algorithm to solve an LPN instance, we make use of the existing techniques and optimize their use. We formalize an LPN algorithm as a path in a graph G and our algorithm is searching for the optimal paths in this graph. Our results bring improvements over the existing work, i.e. we improve the results of the covering code from ASIACRYPT'14 and EUROCRYPT'16. Furthermore, we propose concrete practical codes and a method to find good codes.

1 Introduction

The Learning Parity with Noise (LPN) problem can be seen as a noisy system of linear equations in the binary domain. More specifically, we have a secret s and an adversary that has access to an LPN oracle which provides him tuples of uniformly distributed binary vectors v_i and the inner product between s and v_i to which some noise was added. The noise is represented by a Bernoulli variable with a probability τ to be 1. The goal of the adversary is to recover the secret s. The LPN problem is a particular case of the well-known Learning with Errors (LWE) [34] problem where instead of working in $\mathbb{Z}_2$ we extend the work to a ring $\mathbb{Z}_q$.

The LPN problem is attractive as it is believed to be resistant to quantum computers. Thus, it can be a good candidate for replacing the number-theoretic problems such as factorization and discrete logarithm (which can be easily broken by a quantum algorithm). Also, given its structure, it can be implemented in lightweight devices. The LPN problem is used in the design of the *HB*-family of authentication protocols [10,20,24,25,27,31] and several cryptosystems base their security on its hardness [1,15,16,17,21,26].

Previous Work. LPN is believed to be hard. So far, there is no reduction from hard lattice problems to certify the hardness (like in the case of LWE). Thus, the best way to assess its hardness is by trying to design and improve algorithms that solve it. Over the years, the LPN problem was analyzed and there exist several solving algorithms. The first algorithm to target LPN is the BKW algorithm [6]. This algorithm can be described as a Gaussian elimination on blocks of bits (instead on single bits) where the secret is recovered bit by bit. Several improvements appeared afterwards [19,29]. One idea that improves the algorithm is the use of the fast Walsh-Hadamard transform as we can recover several bits of the secret at once. In their work, Levieil and Fouque [29] provide an analysis with the level of security achieved by different LPN instances and propose secure parameters. Using BKW as a black-box, Lyubashevsky [30] presents an LPN solving algorithm useful for the case when the number of queries is restricted to an adversary. The best algorithm to solve LPN was presented at ASIACRYPT'14 [23] and it introduces the use of the covering codes to improve the performance. Some problems in the computation of complexities were reported [7,37]. As discussed by Bogos et al. [7] and in the ASIACRYPT presentation [23] [1] , the authors used a too optimistic approximation for the bias introduced by their new reduction method, the covering codes. Some complexity terms are further missing (as discussed in Section 2.2) or are not in bit operations. Also, no method to construct covering codes were suggested. At EUROCRYPT'16, Zhang et al. [37] proposed a way to construct good codes by concatenating perfect codes and improved the algorithms. However, some other problem in complexities were reported [9]. The new $\mathsf{LF}(4)$ reduction technique introduced by Zhang et al. [37] was also shown to be incorrect [9].

For the case when the secret is sparse, i.e. its Hamming weight is small, the classical Gaussian elimination proves to give better results [7,8,11].

The LPN algorithms consist of two parts: one in which the size of the secret is reduced and one in which part of the secret is recovered. Once a part of the secret is recovered, the queries are updated and the algorithm restarts to recover the rest of the secret. When trying to recover a secret s of k bits, it is assumed that k can be written as $a \cdot b$, for $a, b \in \mathbb{N}$ (i.e. secret s can be seen as a blocks of b bits). Usually all the reduction steps reduce the size by b bits and the solving algorithm recovers b bits. While the use of the same parameter, i.e. b, for all the operations may be convenient for the implementation, we search for an algorithm that may use different values for each reduction step. We discover that small variations from the fixed b can bring important improvements in the time complexity of the whole algorithm.

Our Contribution. In this work we first *analyze the existing* LPN *algorithms and study the operations that are used in order to reduce the size of the secret. We adjust the expressions of the complexities of each step* (as in some works they were underestimated in the literature). For instance, the results from Guo et

[1] http://des.cse.nsysu.edu.tw/asiacrypt2014/doc/1-1_SolvingLPNUsingCoveringCodes.pdf

al. [23] and Zhang et al. [37] are displayed with corrections in Table 1.[2] (Details for this computation are provided as an additional material for this paper.)

(k,τ)	ASIACRYPT'14 [23]	EUROCRYPT'16 [37]	our results
$(512, 0.125)$	$2^{86.96}(2^{79.9})$ (proceedings) $2^{81.90}(2^{79.7})$ (presentation) [1]	$2^{80.09}(2^{74.73})$	$2^{78.84}$
$(532, 0.125)$	$2^{88.62}(2^{81.82})$	$2^{82.17}(2^{76.90})$	$2^{81.02}$
$(592, 0.125)$	$2^{97.71}(2^{88.07})$	$2^{89.32}(2^{83.84})$	$2^{87.57}$

Table 1: Time complexity to solve LPN (in bit operations). These complexities are based on the formulas from our paper with the most favorable covering codes we constructed from our pool, with adjusted data complexity to reach a failure probability bounded by 33%. Originally claimed complexities by [23] and [37] are under parentheses.

Second, we *improve the theory behind the covering code reduction and show the link with perfect and quasi-perfect codes.* Using the average bias of covering codes allows us to use arbitrary codes and even random ones. Using the algorithm to construct optimal concatenated codes based on a pool of elementary ones allows us to improve complexities. (In Guo et al. [23], only a hypothetical code was assumed to be close to a perfect code; in Zhang et al. [37], only the concatenation of perfect codes are used; in Table 1, our computed complexities are based on the real codes that we built with our bigger pool to have a fair comparison.)

Third, we *optimize the order and the parameters used by the operations that reduce the size of the secret such that we minimize the time complexity required*. We *design a "meta-algorithm" that combines the reduction steps and finds the optimal strategy to solve* LPN. We *automatize the process of finding* LPN *solving algorithms, i.e. given a random* LPN *instance, our algorithm provides the description of the steps that optimize the time complexity*. In our formalization we call such algorithms "optimal chains". We perform a security analysis of LPN based on the results obtained by our algorithm and compare our results with the existing ones. We discover that we improve the complexity compared with the existing results [7,29,37,23], as shown in Table 1.

[2] As for [37], we only reported the results based on LF2 which are better than with LF1, as the LF(4) operation is incorrect [9].

Preliminaries & Notations. Given a domain $\mathcal{D}$, we denote by $x \xleftarrow{U} \mathcal{D}$ the fact that x is drawn uniformly at random from $\mathcal{D}$. By Ber_τ we denote the Bernoulli distribution with parameter τ. By Ber_τ^k we denote the binomial distribution with parameters k and τ. Let $\langle \cdot, \cdot \rangle$ denote the inner product, $\mathbb{Z}_2 = \{0,1\}$ and $\oplus$ denote the bitwise XOR. The Hamming weight of a vector v is denoted by $\mathsf{HW}(v)$.

Organization. In Section 2 we formally define the LPN problem and describe the main tools used to solve it. We carefully analyze the complexity of each step and show in footnote where it differs from the existing literature. Section 3 studies the failure probability of the entire algorithm and validates the use of the average bias in the analysis. Section 4 introduces the bias computation for perfect and quasi-perfect codes. We provide an algorithm to find good codes. The algorithm that searches the optimal strategy to solve LPN is presented in Sections 5 and 6. We illustrate and compare our results in Section 7 and conclude in Section 8. We put in additional material details of our results: the complete list of the chains we obtain (for Table 3 and Table 4), an example of complete solving algorithm, the random codes that we use for the covering code reduction, and an analysis of the results from [23] and [37] to obtain Table 1.

2 LPN

2.1 LPN Definition

The LPN problem can be seen as a noisy system of equations in $\mathbb{Z}_2$ where one is asked to recover the unknown variables. Below, we present the formal definition.

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 $D_{s,\tau}$ the distribution defined as*

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

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

Definition 2 (Search LPN problem). *Given access to an LPN oracle $\mathcal{O}_{s,\tau}^{\mathsf{LPN}}$, 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*

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

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

Remark that we consider here the problem of recovering only a part of the secret. Throughout the literature this is how the LPN problem is formulated. The reason for doing so is that the recovery of the first k' bits dominates the overall complexity. Once we recover part of the secret, the new problem of recovering a shorter secret of $k-k'$ bits is easier.

The LPN problem has a decisional form where one has to distinguish between random vectors of size $k+1$ and the samples from the LPN oracle. In this paper we are interested only in finding algorithms for the search version.

We define $\delta = 1-2\tau$. We call δ the bias of the error bit d. We have $\delta = E((-1)^d)$, with $E(\cdot)$ the expected value. We denote the bias of the secret bits by δ_s. As s is a uniformly distributed random vector, at the beginning we have $\delta_s = 0$.

2.2 Reduction and Solving Techniques

Depending on how many queries are given from the LPN oracle, the LPN solving algorithms are split in 3 categories. With a *linear number of queries*, the best algorithms are exponential, i.e. with $n = \Theta(k)$ the secret is recovered in $2^{\Theta(k)}$ time [32,36]. Given a *polynomial number of queries* $n = k^{1+\eta}$, with $\eta > 0$, one can solve LPN with a sub-exponential time complexity of $2^{O(\frac{k}{\log\log k})}$ [30]. When $\tau = \frac{1}{\sqrt{k}}$ we can improve this result and have a complexity of $e^{\frac{1}{2}\sqrt{k}(\ln k)^2 + O(\sqrt{k}\ln k)}$ [8]. The complexity improves but remains in the sub-exponential range with a *sub-exponential number of queries*. For this category, we have the BKW [6], LF1, LF2 [29], FMICM [19] and the covering code algorithm [23,37]. All these algorithms solve LPN with a time complexity of $2^{O(\frac{k}{\log k})}$ and require $2^{O(\frac{k}{\log k})}$ queries. In the special case when the noise is sparse, a simple Gaussian elimination can be used for the recovery of the secret [7,11]. LF2, covering code or the Gaussian elimination prove to be the best one, depending on the noise level [7].

All these algorithms have a common structure: given an $\mathsf{LPN}_{k,\tau}$ instance with a secret s, they reduce the original LPN problem to a new LPN problem where the secret s' is of size $k' \leq k$ by applying several *reduction techniques*. Then, they recover s' using a *solving method*. The queries are updated and the process is repeated until the whole secret s is recovered. We present here the list of reduction and solving techniques used in the existing LPN solving algorithms. In the next section, we combine the reduction techniques such that we find the optimal reduction phases for solving different LPN instances.

We assume for all the reduction steps that we start with n queries, that the size of the secret is k, the bias of the secret bits is δ_s and the bias of the noise bits is δ. After applying a reduction step, we will end up with n' queries, size

k' and biases δ' and δ'_s. Note that δ_s averages over all secrets although the algorithm runs with one target secret. As it will be clear below, the complexity of all reduction steps only depends on k, n, and the parameters of the steps but not on the biases. Actually, only the probability of success is concerned with biases. We see in Section 3 that the probability of success of the overall algorithm is not affected by this approach. Actually, we will give a formula to compute a value which approximates the average probability of success over the key based on the average bias.

We have the following reduction steps:

– *sparse-secret* changes the secret distribution. In the formal definition of LPN, we take the secret s to be a random row vector of size k. When other reduction steps or the solving phase depends on the distribution of s, one can transform an LPN instance with a random s to a new one where s has the same distribution as the initial noise, i.e. $s \leftarrow \mathsf{Ber}_\tau^k$. The reduction performs the following steps: from the n queries select k of them: $(v_{i_1}, c_{i_1}), \ldots, (v_{i_k}, c_{i_k})$ where the row vectors v_{i_j}, with $1 \leq j \leq k$, are linearly independent. Construct the matrix M as $M = [v_{i_1}^T \cdots v_{i_k}^T]$ and rewrite the k queries as $sM + d' = c'$, where $d' = (d_{i_1}, \ldots, d_{i_k})$. With the rest of $n-k$ queries we do the following:

$$c'_j = \langle v_j (M^T)^{-1}, c' \rangle \oplus c_j = \langle v_j (M^T)^{-1}, d' \rangle \oplus d_j = \langle v'_j, d' \rangle \oplus d_j$$

We have $n-k$ new queries (v'_j, c'_j) where the secret is now d'. In Guo et al. [23], the authors use an algorithm which is inappropriately called "the four Russians algorithm" [2]. This way, the complexity should be of $O\left(\min_{\chi \in \mathbb{N}} \left(kn' \lceil \frac{k}{\chi} \rceil + k^3 + k\chi 2^\chi\right)\right)$.[3] Instead, the Bernstein algorithm [4] works in $O\left(\frac{n'k^2}{\log_2 k - \log_2 \log_2 k} + k^2\right)$. We use the best of the two, depending on the parameters. Thus, we have:

$$\boxed{\begin{array}{l} \textit{sparse-secret}: k' = k;\ n' = n-k;\ \delta' = \delta;\ \delta'_s = \delta \\ \text{Complexity: } O\left(\min_{\chi \in \mathbb{N}} \left(\frac{n'k^2}{\log_2 k - \log_2 \log_2 k} + k^2, kn' \lceil \frac{k}{\chi} \rceil + k^3 + k\chi 2^\chi\right)\right) \end{array}}$$

– *xor-reduce*(b) was first used by the LF2 algorithm. The queries are grouped in equivalence classes according to the values on b random positions. In each equivalence class, we perform the xoring of every pair of queries. The size of the secret is reduced by b bits and the new bias is δ^2. The expected new number of queries is $E(\sum_{i<j} 1_{v_i \text{ matches } v_j \text{ on the } b\text{-bit block}}) = \frac{n(n-1)}{2^{b+1}}$

[3] but the $k^3 + k\chi 2^\chi$ terms is missing in [23].

which improves previous results[4]. When $n \approx 1+2^{b+1}$, the number of queries are maintained. For $n > 1+2^{b+1}$, the number of queries will increase.

$\mathit{xor\text{-}reduce}(b): k' = k-b;\ n' = \frac{n(n-1)}{2^{b+1}};\ \delta' = \delta^2;\ \delta'_s = \delta_s$
Complexity: $O(k \cdot \max(n,n'))$

– $\mathit{drop\text{-}reduce}(b)$ is a reduction used only by the BKW algorithm. It consists in dropping all the queries that are not 0 on a window of b bits. Again, these b positions are chosen randomly. In average, we expect that half of the queries are 0 on a given position. For b bits, we expect to have $\frac{n}{2^b}$ queries that are 0 on this window. The bias is unaffected and the secret is reduced by b bits.

$\mathit{drop\text{-}reduce}(b): k' = k-b;\ n' = \frac{n}{2^b};\ \delta' = \delta;\ \delta'_s = \delta_s$
Complexity: $O(n(1+\frac{1}{2}+\ldots+\frac{1}{2^{b-1}}))$

The complexity of $n(1+\frac{1}{2}+\ldots+\frac{1}{2^{b-1}}) = O(n)$ comes from the fact that we don't need to check all the b bits: once we find a 1 we don't need to continue and just drop the corresponding query.

– $\mathit{code\text{-}reduce}(k,k',\mathsf{params})$ is a method used by the covering code algorithm presented in ASIACRYPT'14. In order to reduce the size of the secret, one uses a linear code $[k,k']$ (which is defined by params) and approximates the v_i vectors to the nearest codeword g_i. We assume that decoding is done in linear time for the code considered. (For the considered codes, decoding is indeed based on table look-ups.) The noisy inner product becomes:

$$\begin{aligned}\langle v_i,s\rangle \oplus d_i &= \langle g'_i G,s\rangle \oplus \langle v_i-g_i,s\rangle \oplus d_i\\ &= \langle g'_i, sG^T\rangle \oplus \langle v_i-g_i,s\rangle \oplus d_i\\ &= \langle g'_i,s'\rangle \oplus d'_i,\end{aligned}$$

where G is the generator matrix of the code, $g_i = g'_i G$, $s' = sG^T \in \{0,1\}^{k'}$ and $d'_i = \langle v_i - g_i, s\rangle \oplus d_i$. We denote $\mathsf{bc} = E((-1)^{\langle v_i-g_i,s\rangle})$ the bias of $\langle v_i-g_i,s\rangle$. We will see in Section 4 how to construct a $[k,k']$ linear code making bc as large as possible.
Here, bc averages the bias over the secret although s is fixed by $\mathit{sparse\text{-}secret}$. It gives the correct average bias δ over the distribution of the key. We will see that it allows to approximate the expected probability of success of the algorithm.
By this transform, no query is lost.

$\mathit{code\text{-}reduce}(k,k',\mathsf{params}): k';\ n' = n;\ \delta' = \delta \cdot \mathsf{bc}$
δ'_s depends on δ_s and G
Complexity: $O(kn)$

[4] In Bogos et al. [7], the number of queries was approximated to $\frac{\frac{n}{2^b}\left(\frac{n}{2^b}-1\right)}{2}$ which is less favorable.

The way δ'_s is computed is a bit more complicated than for the other types of reductions. However, δ_s only plays a role in the *code-reduce* reduction, and we will not consider algorithms that use more than one *code-reduce* reduction.

It is easy to notice that with each reduction operation the number of queries decreases or the bias is getting smaller. In general, for solving LPN, one tries to lose as few queries as possible while maintaining a large bias. We will study in the next section what is a good combination of using these reductions.

After applying the reduction steps, we assume we are left with an $\mathsf{LPN}_{k',\delta'}$ instance where we have n' queries. The original BKW algorithm was using a final solving technique based on majority decoding. Since the LF2 algorithm, we use a better solving technique based on the Walsh Hadamard Transform (WHT).

WHT recovers a block of the secret by computing the fast Walsh Hadamard transform on the function $f(x) = \sum_i 1_{v_i=x}(-1)^{\langle v_i,s\rangle \oplus d_i}$. The Walsh-Hadamard transform is

$$\hat{f}(\nu) = \sum_x (-1)^{\langle \nu,x\rangle} f(x) = \sum_i (-1)^{\langle v_i,s+\nu\rangle \oplus d_i}$$

For $\nu = s$, we have $\hat{f}(s) = \sum_i (-1)^{d_i}$. For a positive bias, we know that most of the noise bits are set to 0. It is the opposite when the bias is negative. So, $|\hat{f}(s)|$ is large and we suppose it is the largest value in the table of $\hat{f}$. Using again the Chernoff bounds, we need to have $n' = 8\ln(\frac{2^{k'}}{\theta})\delta'^{-2}$ [7] queries in order to bound the probability of guessing wrongly the k'-bit secret by θ. We can improve further by applying directly the Central Limit Theorem and obtain a heuristic bound $\varphi(-\sqrt{\frac{n'}{2\delta'^{-2}-1}}) \leq 1-(1-\theta)^{\frac{1}{2^{k'}-1}}$, where $\varphi(x) = \frac{1}{2} + \frac{1}{2}\mathsf{erf}(\frac{x}{\sqrt{2}})$ and erf is the Gauss error function. We obtain that

$$\sqrt{n'} \geq -\sqrt{2\delta'^{-2}-1} \cdot \varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^{k'}-1}}\right). \tag{1}$$

We can derive the approximation of Selçuk [35] that $n' \geq 4\ln(\frac{2^{k'}}{\theta})\delta'^{-2}$. We give the details of our results in Section 3. Complexity of the $\mathsf{WHT}(k')$ is $O(k'2^{k'}\frac{\log_2 n'+1}{2} + k'n')$ as we use the fast Walsh Hadamard Transform[5] [6].

[5] The second term $k'n'$ illustrates the cost of constructing the function f. In cases where $n' > 2^{k'}$ this is the dominant term and it should not be ignored. This was missing in several works [23,7]. For the instance $\mathsf{LPN}_{592,0.125}$ from Guo et al. [23] this makes a big difference as $k' = 64$ and $n' = 2^{69}$; the complexity of WHT with the second term is 2^{75} vs 2^{70} [23]. Given that is must be repeated 2^{13} (as 35 bits of the secret are guessed), the cost of WHT is 2^{88}.

[6] Normally, the values $\hat{f}(\nu)$ have an order of magnitude of $\sqrt{n'}$ so we have $\frac{1}{2}\log_2 n'$ bits.

$\mathsf{WHT}(k')$;

Requires $\sqrt{n'} \geq -\sqrt{2\delta'^{-2}-1} \cdot \varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^{k'}-1}}\right)$

Complexity: $O(k' 2^{k'} \frac{\log_2 n'+1}{2} + k'n')$

Given the reduction and the solving techniques, an $\mathsf{LPN}_{k,\tau}$ solving algorithm runs like this: we start with a k-bit secret and with n queries from the LPN oracle. We reduce the size of the secret by applying several reduction steps and we end up with n' queries where the secret has size k'. We use one solving method, e.g. the WHT, and recover the k'-bit secret with a probability of failure bounded by θ. We chose $\theta = \frac{1}{3}$. We have recovered a part of the secret. To fully recover the whole secret, we update the queries and start another chain to recover more bits, and so on until the remaining $k-k'$ bits are found. For the second part of the secret we will require for the failure probability to be θ^2 and for the i^{th} part it will be θ^i. Thus, if we recover the whole secret in i iterations, the total failure probability will be bounded by $\theta + \theta^2 + \cdots + \theta^i$. Given that we take $\theta = \frac{1}{3}$, we recover the whole secret with a success probability larger than 50%. Experience shows that the time complexity for the first iteration dominates the total complexity.

As we can see in the formulas of each possible step, the computations of k', n', and of the complexity do not depend on the secret weight. Furthermore, the computation of biases is always linear. So, the correct average bias (over the distribution of the key made by the *sparse-secret* transform) is computed. Only the computation of the success probability is non-linear but we discuss about this in the next section. As it only matters in WHT, we will see in Section 3 that the approximation is justified.

3 On Approximating the Probability of Success

Approximating n by using Central Limit Theorem. In order to approximate the number of queries needed to solve the LPN instance we consider when the Walsh Hadamard Transform fails to give the correct secret. We first assume that the bias is positive. We have a failure when for another $\bar{s} \neq s$, we have that $\hat{f}(\bar{s}) > \hat{f}(s)$. Following the analysis from [7], we let $y = A'\bar{s}^T + c'^T$ and $d' = A's^T + c'^T$. We have $\hat{f}(\bar{s}) = \sum_i (-1)^{y_i} = n' - 2.\mathsf{HW}(y)$ and similarly, $\hat{f}(s) = n' - 2.\mathsf{HW}(d')$. So, $\hat{f}(\bar{s}) > \hat{f}(s)$ translates to $\mathsf{HW}(y) \leq \mathsf{HW}(d')$. Therefore

$$\Pr[\hat{f}(\bar{s}) > \hat{f}(s)] = \Pr\left[\sum_{i=1}^{n'} (y_i - d'_i) \leq 0\right].$$

For each $\bar{s}$, we take y as a uniformly distributed random vector and we let $\delta'(s)$ be the bias introduce with a fixed s for d'_i (we recall that our analysis computes

$\delta' = E(\delta'(s))$ over the distribution of s). Let $X_1, \ldots, X_{n'}$ be random variable corresponding to $X_i = y_i - d'_i$. Since $E(y_i) = \frac{1}{2}$, $E(d'_i) = \frac{1}{2} - \frac{\delta'(s)}{2}$ and y_i and d'_i are independent, we have that $E(X_i) = \frac{\delta'(s)}{2}$ and $\mathsf{Var}(X_i) = \frac{2-\delta'(s)^2}{4}$. By using the Central Limit Theorem we obtain that

$$\Pr[X_1 + \ldots + X_{n'} \leq 0] \approx \varphi(Z(s)) \text{ with } Z(s) = -\frac{\delta'(s)}{\sqrt{2-\delta'(s)^2}}\sqrt{n'}$$

where φ can be calculated by $\varphi(x) = \frac{1}{2} + \frac{1}{2}\mathsf{erf}(\frac{x}{\sqrt{2}})$ and erf is the Gauss error function. For $\delta'(s) < 0$, the same analysis with $\hat{f}(\bar{s}) < \hat{f}(s)$ gives the same result. Applying the reasoning for any $s' \neq s$ we obtain that the failure probability is

$$p(s) = 1 - (1-\varphi(Z(s)))^{2^{k'}-1}, \text{if } \delta'(s) > 0$$
$$\text{and } p(s) = 1 - \frac{1}{2^{k'}}, \text{if } \delta'(s) \leq 0.$$

We deduce the following (for $\theta < \frac{1}{2}$)

$$p(s) \leq \theta \Leftrightarrow \sqrt{n'} \geq -\sqrt{2\delta'(s)^{-2}-1}\,\varphi^{-1}\left(1-(1-\theta)^{\frac{1}{2^{k'}-1}}\right) \text{ and } \delta'(s) > 0$$

As a condition for our WHT step, we adopt the inequality in which we replace $\delta'(s)$ by δ'. We give a heuristic argument below to show that it implies $E(p(s)) \leq \theta$, which is what we want.

Note that if we use the approximation $\varphi(Z) \approx -\frac{1}{Z\sqrt{2\pi}}e^{-\frac{Z^2}{2}}$ for $Z \to -\infty$, we obtain the condition $n' \geq 2(2\delta'^{-2}-1)\ln(\frac{2^{k'}-1}{\theta})$. So, our analysis brings an improvement of factor two over the Hoeffding bound method used by Bogos et al. [7] that requires $n' \geq 8\delta'^{-2}\ln(\frac{2^{k'}}{\theta})$.

On the validity of the using the bias average. The above computation is correct when using $\delta'(s)$ but we use $\delta' = E(\delta'(s))$ instead. If no *code-reduce* step is used, $\delta'(s)$ does not depend on s and we do have $\delta'(s) = \delta'$. However, when a *code-reduce* is used, the bias depends on the secret which is obtained after the *sparse-secret* step. For simplicity, we let s denote this secret. The bias $\delta'(s)$ is actually of form $\delta'(s) = \delta^{2^x}\mathsf{bc}(s)$ where x is the number of *xor-reduce* steps and $\mathsf{bc}(s)$ is the bias introduced by *code-reduce* depending on s. The values of $\delta'(s)$, $Z(s)$, and $p(s)$ are already defined above. We define $Z = -\frac{\delta'}{\sqrt{2-\delta'^2}}\sqrt{n'}$ and $p = 1-(1-\varphi(Z))^{2^{k'}-1}$. Clearly, $E(p(s))$ is the average failure probability over the distribution of the secret obtained after *sparse-secret*.

Our method ensures that $\delta' = E(\delta'(s))$ over the distribution of s. Since δ' is typically small (after a few *xor-reduce* steps, δ^{2^x} is indeed very small), we can consider $Z(s)$ as a linear function of $\delta'(s)$ and have $E(Z(s)) \approx Z$. This is confirmed by experiment. We make the **heuristic approximation** that

$$E\left(1-(1-\varphi(Z(s)))^{2^{k'}-1}\right) \approx 1-(1-\varphi(E(Z(s))))^{2^{k'}-1} \approx 1-(1-\varphi(Z))^{2^{k'}-1}$$

So, $E(p(s)) \approx p$.[7]

We did some experiments based on some examples in order to validate our heuristic assumption. Our results show indeed that $E(Z(s)) \approx Z$. There is a small gap between $E(p(s))$ and p but this does not affect our results. Actually, we are in a phase transition region so any tiny change in the value of n' makes $E(p(s))$ change a lot. We include our results in the additional material. Thus, ensuring that $p \leq \theta$ with the above analysis based on the average bias ensures that the expected failure probability to be bounded by θ.

We also observed that the reduction *code-reduce* can introduce problems. More precisely, what can go wrong is that s can have, with a given probability, a negative $\delta'(s)$ bias or a component in one of the concatenated codes giving a zero bias, making WHT to fail miserably.

4 Bias of the Code Reduction

In this section we present how to compute the bias introduced by a *code-reduce*. Recall that the reduction *code-reduce*(k,k') introduces a new noise:

$$\langle v_i, s\rangle \oplus d_i = \langle g'_i, s'\rangle \oplus \langle v_i - g_i, s\rangle \oplus d_i,$$

where $g_i = g'_i G$ is the nearest codeword of v_i and $s' = sG^T$. Note that g_i is not necessarily unique, specially if the code is not perfect. We take $g_i = \mathsf{Decode}(v_i)$ obtained from an arbitrary decoding algorithm. Then the noise bc can be computed by the following formula:

$$\begin{aligned} \mathsf{bc} = E((-1)^{\langle v_i - g_i, s\rangle}) &= \sum_{e\in\{0,1\}^k} \Pr[v_i - g_i = e] E((-1)^{\langle e,s\rangle}) \\ &= \sum_{w=0}^{k} \sum_{\substack{e\in\{0,1\}^k,\\ \mathsf{HW}(e)=w}} \Pr[v_i - g_i = e]\delta_s^w = E\left(\delta_s^{\mathsf{HW}(v_i-g_i)}\right) \end{aligned}$$

[7] Note that Zhang et al. [37] implicitly does the same assumption as they use the average bias as well.

for a δ_s-sparse secret. (We recall that the *sparse-secret* reduction step randomizes the secret.) So, the probability space is over the distribution of v_i *and* the distribution of s. Later, we consider $\mathsf{bc}(s) = E((-1)^{\langle v_i - g_i, s\rangle})$ over the distribution over v_i only. (In the work of Guo et al. [23], only $\mathsf{bc}(s)$ is considered. In Zhang et al. [37], our bc was also considered.) In the last expression of bc, we see that the ambiguity in decoding does not affect bc as long as the Hamming distance $\mathsf{HW}(v_i - \mathsf{Decode}(v_i))$ is not ambiguous. This is a big advantage of averaging in bc as it allows to use non-perfect codes. From this formula, we can see that the decoding algorithm $v_i \to g_i$ making $\mathsf{HW}(v_i - g_i)$ minimal makes bc maximal. In this case, we obtain

$$\mathsf{bc} = E\left(\delta_s^{d(v_i, C)}\right), \tag{2}$$

where C is the code and $d(v_i, C)$ denotes the Hamming distance of v_i from C.

For a code C, the *covering radius* is $\rho = \max_v d(v, C)$. The *packing radius* is the largest radius R such that the balls of this radius centered on all codewords are non-overlapping. So, the packing radius is $R = \left\lfloor \frac{D-1}{2} \right\rfloor$ where D is the minimal distance. We further have $\rho \geq \left\lfloor \frac{D-1}{2} \right\rfloor$. A *perfect code* is characterized by $\rho = \left\lfloor \frac{D-1}{2} \right\rfloor$. A *quasi-perfect code* is characterized by $\rho = \left\lfloor \frac{D-1}{2} \right\rfloor + 1$.

Theorem 1. *We consider a $[k, k', D]$ linear code C, where k is the length, k' is the dimension, and D is the minimal distance. For any integer r and any positive bias δ_s, we have*

$$\mathsf{bc} \leq 2^{k'-k} \sum_{w=0}^{r} \binom{k}{w} (\delta_s^w - \delta_s^{r+1}) + \delta_s^{r+1}$$

where bc is a function of δ_s defined by (2). Equality for any δ_s such that $0 < \delta_s < 1$ implies that C is perfect or quasi-perfect. In that case, the equality is reached when taking the packing radius $r = R = \left\lfloor \frac{D-1}{2} \right\rfloor$.

By taking r as the largest integer such that $\sum_{w=0}^{r} \binom{k}{w} \leq 2^{k-k'}$ (which is the packing radius $R = \left\lfloor \frac{D-1}{2} \right\rfloor$ for perfect and quasi-perfect codes), we can see that if a perfect $[k, k']$ code exists, it makes bc maximal. Otherwise, if a quasi-perfect $[k, k']$ code exists, it makes bc maximal.

Proof. Let decode be an optimal deterministic decoding algorithm. The formula gives us that

$$\mathsf{bc} = 2^{-k} \sum_{g \in C} \sum_{v \in \mathsf{decode}^{-1}(g)} \delta_s^{\mathsf{HW}(v-g)}$$

We define $\mathsf{decode}_w^{-1}(g) = \{v \in \mathsf{decode}^{-1}(g); \mathsf{HW}(v-g) = w\}$ and $\mathsf{decode}_{>r}^{-1}(g)$ the union of all $\mathsf{decode}_w^{-1}(g)$ for $w > r$. For all r, we have

$$\begin{aligned}
&\sum_{v \in \mathsf{decode}^{-1}(g)} \delta_s^{\mathsf{HW}(v-g)} \\
&= \sum_{w=0}^{r} \binom{k}{w} \delta_s^w + \sum_{w=0}^{r} \left(\#\mathsf{decode}_w^{-1}(g) - \binom{k}{w} \right) \delta_s^w + \sum_{w>r} \delta_s^w \#\mathsf{decode}_w^{-1}(g) \\
&\leq \sum_{w=0}^{r} \binom{k}{w} \delta_s^w + \sum_{w=0}^{r} \left(\#\mathsf{decode}_w^{-1}(g) - \binom{k}{w} \right) \delta_s^w + \delta_s^{r+1} \#\mathsf{decode}_{>r}^{-1}(g) \\
&\leq \sum_{w=0}^{r} \binom{k}{w} \delta_s^w + \delta_s^{r+1} \left(\#\mathsf{decode}^{-1}(g) - \sum_{w=0}^{r} \binom{k}{w} \right)
\end{aligned}$$

where we used $\delta_s^w \leq \delta_s^{r+1}$ for $w > r$, $\#\mathsf{decode}_w^{-1}(g) \leq \binom{k}{w}$ and $\delta_s^w \geq \delta_s^{r+1}$ for $w \leq r$. We further have equality if and only if the ball centered on g of radius r is included in $\mathsf{decode}^{-1}(g)$ and the ball of radius $r+1$ contains $\mathsf{decode}^{-1}(g)$. By summing over all $g \in C$, we obtain the result.

So, the equality case implies that the packing radius is at least r and the covering radius is at most $r+1$. Hence, the code is perfect or quasi-perfect. Conversely, if the code is perfect or quasi-perfect and r is the packing radius, we do have equality. □

So, for quasi-perfect codes, we can compute

$$\mathsf{bc} = 2^{k'-k} \sum_{w=0}^{R} \binom{k}{w} (\delta_s^w - \delta_s^{R+1}) + \delta_s^{R+1} \tag{3}$$

with $R = \left\lfloor \frac{D-1}{2} \right\rfloor$. For perfect codes, the formula simplifies to

$$\mathsf{bc} = 2^{k'-k} \sum_{w=0}^{R} \binom{k}{w} \delta_s^w \tag{4}$$

4.1 Bias of a Repetition Code

Given a $[k,1]$ repetition code, the optimal decoding algorithm is the majority decoding. We have $D = k$, $k' = 1$, $R = \left\lfloor \frac{k-1}{2} \right\rfloor$. For k odd, the code is perfect so $\rho = R$. For k even, the code is quasi-perfect so $\rho = R+1$. Using (3) we obtain

$$\mathsf{bc} = \begin{cases} \sum_{w=0}^{\frac{k-1}{2}} \frac{1}{2^{k-1}} \binom{k}{w} \delta_s^w & \text{if } k \text{ is odd} \\ \\ \sum_{w=0}^{\frac{k}{2}-1} \frac{1}{2^{k-1}} \binom{k}{w} \delta_s^w + \frac{1}{2^k} \binom{k}{k/2} \delta_s^{\frac{k}{2}} & \text{if } k \text{ is even} \end{cases}$$

We give below the biases obtained for some $[k,1]$ repetition codes.

$[k,1]$	bias
$[2,1]$	$\frac{1}{2}\delta_s + \frac{1}{2}$
$[3,1]$	$\frac{3}{4}\delta_s + \frac{1}{4}$
$[4,1]$	$\frac{3}{8}\delta_s^2 + \frac{1}{2}\delta_s + \frac{1}{8}$
$[5,1]$	$\frac{5}{8}\delta_s^2 + \frac{5}{16}\delta_s + \frac{1}{16}$
$[6,1]$	$\frac{5}{16}\delta_s^3 + \frac{15}{32}\delta_s^2 + \frac{3}{16}\delta_s + \frac{1}{32}$
$[7,1]$	$\frac{35}{64}\delta_s^3 + \frac{21}{64}\delta_s^2 + \frac{7}{64}\delta_s + \frac{1}{64}$
$[8,1]$	$\frac{35}{128}\delta_s^4 + \frac{7}{16}\delta_s^3 + \frac{7}{32}\delta_s^2 + \frac{1}{16}\delta_s + \frac{1}{128}$
$[9,1]$	$\frac{63}{128}\delta_s^4 + \frac{21}{64}\delta_s^3 + \frac{9}{64}\delta_s^2 + \frac{9}{256}\delta_s + \frac{1}{256}$
$[10,1]$	$\frac{63}{256}\delta_s^5 + \frac{105}{256}\delta_s^4 + \frac{15}{64}\delta_s^3 + \frac{45}{512}\delta_s^2 + \frac{5}{256}\delta_s + \frac{1}{512}$

4.2 Bias of a Perfect Code

In previous work [23,37], the authors assume a perfect code. In this case, $\sum_{w=0}^{R}\binom{k}{w} = 2^{k-k'}$ and we can use (4) to compute bc. There are not so many binary linear codes which are perfect. Except the repetition codes with odd length, the only ones are the trivial codes $[k,k,1]$ with $R = \rho = 0$ and $\mathsf{bc} = 1$, the Hamming codes $[2^\ell - 1, 2^\ell - \ell - 1, 3]$ for $\ell \geq 2$ with $R = \rho = 1$, and the Golay code $[23,12,7]$ with $R = \rho = 3$.

For the Hamming codes, we have

$$\mathsf{bc} = 2^{-\ell}\sum_{w=0}^{1}\binom{2^\ell - 1}{w}\delta_s^w = \frac{1 + (2^\ell - 1)\delta_s}{2^\ell}$$

For the Golay code, we obtain

$$\mathsf{bc} = 2^{-11}\sum_{w=0}^{3}\binom{23}{w}\delta_s^w = \frac{1 + 23\delta_s + 253\delta_s^2 + 1771\delta_s^3}{2^{11}}$$

Formulae (2), (3), (4) for bc are new. Previously [7,23], the value bc_w of $\mathsf{bc}(s)$ for any s of Hamming weight w was approximated to

$$\mathsf{bc}_w = 1 - 2\frac{1}{S(k,\rho)}\sum_{\substack{i \leq \rho, \\ i \text{ odd}}}\binom{w}{i}S(k-w, \rho - i),$$

where w is the Hamming weight of the k-bit secret and $S(k',\rho)$ is the number of k'-bit strings with weight at most ρ. Intuitively the formula counts the number of $v_i - g_i$ that produce an odd number of xor with the 1's of the secret. (See [7,23].) So, Guo et al. [23] assumes a fixed value for the weight w of the secret and considers the probability that w is not correct. If w is lower, the actual bias is larger but if w is larger, the computed bias is overestimated and the algorithm fails.

For instance, with a $[3,1]$ repetition code, the correct bias is $\mathsf{bc} = \frac{3}{4}\delta_s + \frac{1}{4}$ following our formula. With a fixed w, it is of $\mathsf{bc}_w = 1 - \frac{w}{2}$ [7,23]. The probability of w to be correct is $\binom{k}{w}\tau^w(1-\tau)^{k-w}$. We take the example of $\tau = \frac{1}{3}$ so that $\delta_s = \frac{1}{3}$.

w	bc_w	$\Pr[w]$	$\Pr[w], \tau = \frac{1}{3}$
0	1	$(1-\tau)^3$	0.2963
1	$\frac{1}{2}$	$3\tau(1-\tau)^2$	0.4444
2	0	$3\tau^2(1-\tau)$	0.2222
3	$-\frac{1}{2}$	τ^3	0.0370

So, by taking $w = 1$, we have $\delta = \mathsf{bc}_w = \frac{1}{2}$ but the probability of failure is about $\frac{1}{4}$. Our approach uses the average bias $\delta = \mathsf{bc} = \frac{1}{2}$.

4.3 Using Quasi-Perfect Codes

If C' is a $[k-1,k',D]$ perfect code with $k' > 1$ and if there exists some codewords of odd length, we can extend C', i.e., add a parity bit and obtain a $[k,k']$ code C. Clearly, the packing radius of C is at least $\left\lfloor \frac{D-1}{2} \right\rfloor$ and the covering radius is at most $\left\lfloor \frac{D-1}{2} \right\rfloor + 1$. For $k' > 1$, there is up to one possible length for making a perfect code of dimension k'. So, C is a quasi-perfect, its packing radius is $\left\lfloor \frac{D-1}{2} \right\rfloor$ and its covering radius is $\left\lfloor \frac{D-1}{2} \right\rfloor + 1$.

If C' is a $[k+1,k',D]$ perfect code with $k' > 1$, we can puncture it, i.e., remove one coordinate by removing one column from the generating matrix. If we chose to remove a column which does not modify the rank k', we obtain a $[k,k']$ code C. Clearly, the packing radius of C is at least $\left\lfloor \frac{D-1}{2} \right\rfloor - 1$ and the covering radius is at most $\left\lfloor \frac{D-1}{2} \right\rfloor$. For $k' > 1$, there is up to one possible length for making a perfect code of dimension k'. So, C is a quasi-perfect, its packing radius is $\left\lfloor \frac{D-1}{2} \right\rfloor - 1$ and its covering radius is $\left\lfloor \frac{D-1}{2} \right\rfloor$.

Hence, we can use extended Hamming codes $[2^\ell, 2^\ell - \ell - 1]$ with packing radius 1 for $\ell \geq 3$, punctured Hamming codes $[2^\ell - 2, 2^\ell - \ell - 1]$ with packing radius 0 for $\ell \geq 3$, the extended Golay code $[24,12]$ with packing radius 3, and the punctured Golay code $[22,12]$ with packing radius 2.

There actually exist many constructions for quasi-perfect linear binary codes. We list a few in Table 2. We took codes listed in the existing literature [14, Table 1], [33, p. 122], [22, p. 47], [18, Table 1], [13, p. 313], and [3, Table I]. In Table 2, k, k', D, and R denote the length, the dimension, the minimal distance, and the packing radius, respectively.

Table 2: Perfect and Quasi-Perfect Binary Linear Codes

name	type	$[k,k',D]$	R	comment	ref.
	P	$[k,k,1]$, $k \geq 1$	0	$[*,\ldots,*]$	
r	P	$[k,1,k]$, k odd	$\frac{k-1}{2}$	repetition code	
H	P	$[2^\ell-1, 2^\ell-\ell-1, 3]$, $\ell \geq 3$,	1	Hamming code	
G	P	$[23,12,7]$	3	Golay code	
	QP	$[k,k-1,1]$	0	$[*,\ldots,*,0]$	
r	QP	$[k,1,k]$, k even	$\frac{k}{2}-1$	repetition code	
eG	QP	$[24,12,8]$	3	extended Golay code	
pG	QP	$[22,12,6]$	2	punctured Golay code	
eH	QP	$[2^\ell, 2^\ell-\ell-1, 4]$, $\ell \geq 2$	1	extended Hamming code	
	QP	$[2^\ell-1, 2^\ell-\ell, 1]$, $\ell \geq 2$,	0	Hamming with an extra word	
pH	QP	$[2^\ell-2, 2^\ell-\ell-1, 2]$, $\ell \geq 2$	0	punctured Hamming	
HxH	QP	$[2*(2^\ell-1), 2*(2^\ell-\ell-1)]$, $\ell \geq 2$	1	Hamming $\times$ Hamming	[14]
upack	QP	$[2^\ell-2, 2^\ell-\ell-2, 3]$, $\ell \geq 3$	1	uniformly packed	[14]
2BCH	QP	$[2^\ell-1, (2^\ell-1)-(2*\ell)]$, $\ell \geq 3$	2	2-e.c. BCH	[14]
Z	QP	$[2^\ell+1, (2^\ell+1)-(2*\ell)]$, $\ell > 3$ even	2	Zetterberg	[14]
rGop	QP	$[2^\ell-2, (2^\ell-2)-(2*\ell)]$, $\ell > 3$ even	2	red. Goppa	[14]
iGop	QP	$[2^\ell, (2^\ell)-(2*\ell)]$, $\ell > 2$ odd	2	irred. Goppa	[14]
Mclas	QP	$[2^\ell-1, (2^\ell-1)-2*\ell]$, $\ell > 2$ odd	2	Mclas	[14]
S	QP	$[5,2]$, $[9,5]$, $[10,5]$, $[11,6]$	1	Slepian	[33]
S	QP	$[11,4]$	2	Slepian	[33]
FP	QP	$[15,9]$, $[21,14]$, $[22,15]$, $[23,16]$	1	Fontaine-Peterson	[33]
W	QP	$[19,10]$, $[20,11]$, $[20,13]$, $[23,14]$	2	Wagner	[33]
P	QP	$[21,12]$	2	Prange	[33]
FP	QP	$[25,12]$	3	Fontaine-Peterson	[33]
W	QP	$[25,15]$, $[26,16]$, $[27,17]$, $[28,18]$, $[29,19]$, $[30,20]$, $[31,20]$	1	Wagner	[33]
GS	QP	$[13,7]$, $[19,12]$	1	GS85	[22]
BBD	QP	$[7,3,3]$, $[9,4,4]$, $[10,6,3]$, $[11,7,3]$, $[12,7,3]$, $[12,8,3]$, $[13,8,3]$, $[13,9,3]$, $[14,9,3]$, $[15,10,3]$, $[16,10,3]$, $[17,11,4]$, $[17,12,3]$, $[18,12,4]$, $[18,13,3]$, $[19,13,3]$, $[19,14,3]$, $[20,14,4]$	1	BBD08	[3]
BBD	QP	$[22,13,5]$	2	BBD08	[3]

4.4 Finding the Optimal Concatenated Code

The linear code $[k,k']$ is typically instantiated by a concatenation of elementary codes for practical purposes. By "concatenation" of m codes $C_1,\ldots,C_m$, we mean the code formed by all $g_{i,1}\|\cdots\|g_{i,m}$ obtained by concatenating any set of $g_{i,j}\in C_j$. Decoding $v_1\|\cdots\|v_m$ is based on decoding each $v_{i,j}$ in C_j independently. If all C_j are small, this is done by a table lookup. So, concatenated codes are easy to implement and to decode. For $[k,k']$ we have the concatenation of $[k_1,k'_1],\ldots,[k_m,k'_m]$ codes, where $k_1+\cdots+k_m=k$ and $k'_1+\cdots+k'_m=k'$. Let v_{ij},g_{ij},s'_j denote the j^{th} part of v_i,g_i,s' respectively, corresponding to the concatenated $[k_j,k'_j]$ code. The bias of $\langle v_{ij}-g_{ij},s_j\rangle$ in the code $[k_j,k'_j]$ is denoted by bc_j. As $\langle v_i-g_i,s\rangle$ is the xor of all $\langle v_{ij}-g_{ij},s_j\rangle$, the total bias introduced by this operation is computed as $\mathsf{bc}=\prod_{j=1}^{k'}\mathsf{bc}_j$ and the combination $\mathsf{params}=([k_1,k'_1],\ldots,[k_m,k'_m])$ is chosen such that it gives the highest bias.

The way these params are computed is the following: we start by computing the biases for all elementary codes. I.e. we compute the biases for all codes from Table 2. We may add random codes that we found interesting. (For these, we use (2) to compute bc.)[8] Next, for each $[i,j]$ code we check to see if there is a combination of $[i-n,j-m]$,$[n,m]$ codes that give a better bias, where $[n,m]$ is either a repetition code, a Golay code or a Hamming code. We illustrate below the algorithm to find the optimal concatenated code. This algorithm was independently proposed by Zhang et al. [37] (with perfect codes only).

Algorithm 1 Finding the optimal params and bias

Input: k
Output: table for the optimal bias for each $[i,j]$ code, $1\le j<i\le k$

initialize all $\mathsf{bias}(i,j)=0$
initialize $\mathsf{bias}(1,1)=1$
initialize the bias for all elementary codes
for all $j:2$ to k **do**
  for all $i:j+1$ to k **do**
    for all elementary code $[n,m]$ **do**
      if $|\mathsf{bias}(i-n,j-m)\cdot\mathsf{bias}(n,m)|>|\mathsf{bias}(i,j)|$ **then**
        $\mathsf{bias}(i,j)=\mathsf{bias}(i-n,j-m)\cdot\mathsf{bias}(n,m)$
        $\mathsf{params}(i,j)=\mathsf{params}(i-n,j-m)\cup\mathsf{params}(n,m)$

Using $O(k)$ elementary codes, this procedure takes $O(k^3)$ time and we can store all params for any combination $[i,j]$, $1\le j<i\le k$ with $O(k^2)$ memory.

[8] The random codes that we used are provided as an additional material to this paper.

5 The Graph of Reduction Steps

Having in mind the reduction methods described in Section 2, we formalize an LPN solving algorithm in terms of finding the best chain in a graph. The intuition is the following: in an LPN solving algorithm we can see each reduction step as an edge from a $(k, \log_2 n)$ instance to a new instance $(k', \log_2 n')$ where the secret is smaller, $k' \leq k$, we have more or less number of queries and the noise has a different bias. For example, a *xor-reduce*(b) reduction turns an $(k, \log_2 n)$ instance with bias δ into $(k', \log_2 n')$ with bias δ' where $k' = k - b$, $n' = \frac{n(n-1)}{2^{b+1}}$ and $\delta' = \delta^2$. By this representation, the reduction phase represents a chain in which each edge is a reduction type moving from LPN with parameters (k, n) to LPN with parameters (k', n') and that ends with an instance (k_i, n_i) used to recover the k_i-bit length secret by a solving method. The chain terminates by the fast Walsh-Hadamard solving method.

We formalize the reduction phase as a chain of reduction steps in a graph $G = (V, E)$. The set of vertices V is composed of $V = \{1, \ldots, k\} \times L$ where L is a set of real numbers. For instance, we could take $L = \mathbb{R}$ or $L = \mathbb{N}$. For efficiency reasons, we could even take $L = \{0, \ldots, \eta\}$ for some bound η. Every vertex saves the size of the secret and the logarithmic number of queries; i.e. a vertex $(k, \log_2 n)$ means that we are in an instance where the size of the secret is k and the number of queries available is n. An edge from one vertex to another is given by a reduction step. An edge from $(k, \log_2 n)$ to a $(k', \log_2 n')$ has a label indicating the type of reduction and its parameters (e.g. *xor-reduce*$(k - k')$ or *code-reduce*(k, k', params)). This reduction defines some α and β coefficients such that the bias δ' after reduction is obtained from the bias δ before the reduction by

$$\log_2 \delta'^2 = \alpha \log_2 \delta^2 + \beta$$

where $\alpha, \beta \in \mathbb{R}$.

We denote by $\lceil \lambda \rceil_L$ the smallest element of L which is at least equal to λ and by $\lfloor \lambda \rfloor_L$ the largest element of L which is not larger than λ. In general, we could use a rounding function $\mathsf{Round}_L(\lambda)$ such that $\mathsf{Round}_L(\lambda)$ is in L and approximates λ.

The reduction steps described in Subsection 2.2 can be formalized as follows:

- *sparse-secret*: $(k, \log_2 n) \to (k, \mathsf{Round}_L(\log_2 (n - k)))$ and $\alpha = 0, \beta = 0$
- *xor-reduce*(b): $(k, \log_2 n) \to (k - b, \mathsf{Round}_L(\log_2 \left(\frac{n(n-1)}{2^{b+1}}\right)))$ and $\alpha = 2, \beta = 0$
- *drop-reduce*(b): $(k, \log_2 n) \to (k - b, \mathsf{Round}_L(\log_2 (\frac{n}{2^b})))$ and $\alpha = 1, \beta = 0$

– *code-reduce*(k, k', params): $(k, \log_2 n) \to (k', \log_2 n)$ and $\alpha = 1, \beta = \log_2 \mathsf{bc}^2$, where bc is the bias introduced by the covering code reduction using a $[k, k']$ linear code defined by params.

Below, we give the formal definition of a reduction chain.

Definition 3 (Reduction chain). *Let*

$$\mathcal{R} = \{sparse\text{-}secret, xor\text{-}reduce(b), drop\text{-}reduce(b), code\text{-}reduce(k, k', \mathsf{params})\}$$

for $k, k', b \in \mathbb{N}$. A ***reduction chain*** *is a sequence*

$$(k_0, \log_2 n_0) \xrightarrow{e_1} (k_1, \log_2 n_1) \xrightarrow{e_2} \ldots \xrightarrow{e_i} (k_i, \log_2 n_i),$$

where the change $(k_{j-1}, \log_2 n_{j-1}) \to (k_j, \log_2 n_j)$ is performed by one reduction from $\mathcal{R}$, for all $0 < j \leq i$.

A chain is ***simple*** *if it is accepted by the automaton from Figure 1.*

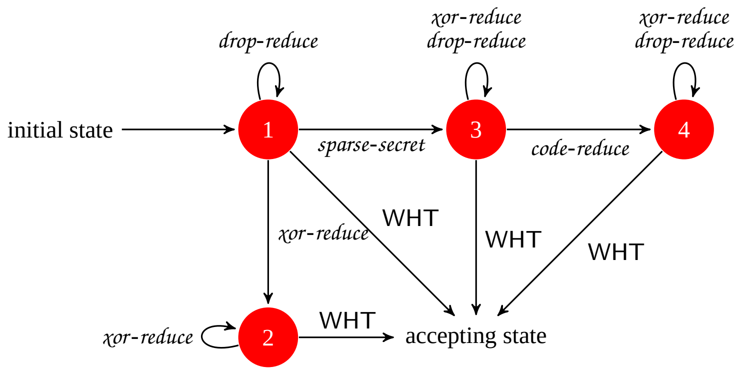

Fig. 1: Automaton accepting simple chains

Remark: Restrictions for simple chains are modelled by the automaton in Figure 1. We restrict to simple chains as they are easier to analyze. Indeed, *sparse-secret* is only used to raise δ_s to make *code-reduce* more effective. And, so far, it is hard to analyze sequences of *code-reduce* steps as the first one may destroy the uniform and high δ_s for the next ones. This is why we exclude multiple *code-reduce* reductions in a simple chain. So, we use up to one *sparse-secret* reduction, always one before *code-reduce*. And *sparse-secret* occurs before δ decreases. For convenience, we will add a state of the automaton to the vertex in V.

Definition 4 (Exact chain). *An* ***exact chain*** *is a simple reduction chain for* $L = \mathbb{R}$. *I.e.* Round_L *is the identity function.*

A chain which is not exact is called **rounded**.

For solving LPN we are interested in those chains that end with a vertex $(k_i, \log_2 n_i)$ which allows to call a WHT solving algorithm to recover the k_i-bit secret. We call these chains valid chains and we define them below.

Definition 5 (Valid reduction chain). *Let*

$$(k_0, \log_2 n_0) \xrightarrow{e_1} (k_1, \log_2 n_1) \xrightarrow{e_2} \cdots \xrightarrow{e_i} (k_i, \log_2 n_i)$$

be a reduction chain with $e_j = (\alpha_j, \beta_j, .)$. *Let* δ_j *be the bias corresponding to the vertex* $(k_j, \log_2 n_j)$ *iteratively defined by* $\delta_0 = \delta$ *and* $\log_2 \delta_j^2 = \alpha_j \log_2 \delta_{j-1}^2 + \beta_j$ *for* $j = 1, \ldots, i$. *We say the chain is a* θ-***valid reduction chain*** *if* n_i *satisfies (1) from page 8 for* $\delta' = \delta_i$ *and* $n' = n_i$.

The *time complexity* of a chain $(e_1, \ldots, e_i)$ is simply the sum of the complexity of each reduction step $e_1, e_2, \ldots, e_i$ and WHT. We further define the **max-complexity** of a chain which is the maximum of the complexity of each reduction step and WHT. The max-complexity is a good approximation of the complexity. Our goal is to find a chain with optimal complexity. What we achieve is that, *given a set L*, we find a *rounded* chain with optimal *max-complexity* up to some given precision.

5.1 Towards Finding the Best LPN Reduction Chain

In this section we present the algorithm that helps finding the optimal valid chains for solving LPN. As aforementioned, we try to find the valid chain with optimal max-complexity for solving an $\mathsf{LPN}_{k,\tau}$ instance in our graph G.

The first step of the algorithm is to construct the directed graph $G = (V, E)$. We take the set of vertices $V = \{1, \ldots, k\} \times L \times \{1, 2, 3, 4\}$ which indicate the size of the secret, the logarithmic number of queries and the state in the automaton in Figure 1. Each edge $e \in E$ represents a reduction step and is labelled with the following information: $(k_1, \log_2 n_1, st) \overset{\alpha,\beta,t}{\rightarrow} (k_2, \log_2 n_2, st')$ where t is one of the reduction steps and α and β save information about how the bias is affected by this reduction step.

The graph has $O(k \cdot |L|)$ vertices and each vertex has $O(k)$ edges. So, the size of the graph is $O(k^2 \cdot |L|)$.

Thus, we construct the graph G with all possible reduction steps and from it we try to see what is the optimal simple rounded chain in terms of max-complexity. We present in Algorithm 2 the procedure to construct the graph G

that contains all possible reduction steps with a time complexity bounded by 2^{η} (As explained below, Algorithm 2 is not really used).

The procedure of finding the optimal valid chain is illustrated in Algorithm 3. The procedure of finding a chain with upper bounded max-complexity is illustrated in Algorithm 4.

Algorithm 2 Construction of graph G

Input: k, τ, L, η
Output: graph $G = (V, E)$ containing all the reduction steps that have a complexity smaller than 2^{η}

$V = \{1, \ldots, k\} \times L \times \{1, \ldots, 4\}$
E is the set of all $((i, \eta_1, st), (j, \eta_2, st'))$ labelled by (α, β, t) such that there is a $st \xrightarrow{t} st'$ transition in the automaton and for
$t =$ *sparse-secret*:
for all $\eta : 1$ such that $\mathsf{lcomp} \leq \eta$ **do** set the edge
    where $i = k$, $(j, \eta_2) = (i, \mathsf{Round}_L(\log_2(2^{\eta_1} - i)))$, $\alpha = 1$, $\beta = 0$, $\mathsf{lcomp} = \min_x \log_2(\frac{(2^{\eta_1} - i)i^2}{log_2 i - log_2 log_2 i} + i^2, i(2^{\eta_1} - i)\lceil \frac{i}{2^x} \rceil + i^3 + i2^{x+2^x})$
$t =$ *xor-reduce*:
for all (i, η_1, b) such that $b \geq 1$ and $\mathsf{lcomp} \leq \eta$ **do** set the edge
    where $(j, \eta_2) = (i - b, \mathsf{Round}_L(\eta_1 - 1 + \log_2(\frac{2^{\eta_1}}{2^b} - 1)))$, $\alpha = 2$, $\beta = 0$, $\mathsf{lcomp} = \log_2 i + \max(\eta_1, \eta_2)$
$t =$ *drop-reduce*:
for all (i, η_1, b) such that $b \geq 1$ and $\mathsf{lcomp} \leq \eta$ **do** set the edge
    where $(j, \eta_2) = (i - b, \mathsf{Round}_L(\eta_1 - b))$, $\alpha = 1$, $\beta = 0$, $\mathsf{lcomp} = \log_2 b + \eta_1$
$t =$ *code-reduce*:
for all (i, η_1, j) such that $j < i$ and $\mathsf{lcomp} \leq \eta$ **do** set the edge
    where $\eta_2 = \eta_1$, $\alpha = 1$, $\beta = \log_2 \mathsf{bc}^2$, $\mathsf{lcomp} = \log_2 i + \eta_1$, bc is the bias from the optimal $[i, j]$ code

---

Algorithm 4 receives as input the parameters k and τ for the LPN instance, the parameter θ which represents the bound on the failure probability in recovering the secret. Parameter η represents an upper bound for the logarithmic complexity of each reduction step. Given η, we build the graph G which contains all possible reductions with time complexity smaller than 2^{η} (Step 4). Note that we don't really call Algorithm 2. Indeed, we don't need to store the edges of the graph. We rather keep a way to enumerate all edges going to a given vertex (in Step 11) by using the rules described in Algorithm 2.

For each vertex, we iteratively define Δ^{st} and Best^{st}, the best reduction step to reach a vertex and the value of the corresponding error bias. The best reduction step is the one that maximizes the bias. We define these values iteratively until we reach a vertex from which the WHT solving algorithm succeeds with

Algorithm 3 Search for a rounded chain with optimal max-complexity

Input: $k, \tau, \theta, \mathsf{precision}$
Output: a valid simple rounded chain in which rounding uses a given precision

set $\mathsf{found} = \mathtt{bruteforce}$ ▷ found is the best found algorithm
set $\mathsf{increment} = k$
set $\eta = k$ ▷ 2^η is a bound on the max-complexity
repeat
  set $\mathsf{increment} \leftarrow \frac{1}{2}\mathsf{increment}$
  define $L = \{0, \mathsf{precision}, 2 \times \mathsf{precision}, \ldots\} \cap [0, \eta - \mathsf{increment}]$
  run $(\mathsf{out}, \mathsf{success}) = \mathsf{Search}(k, \tau, \theta, L, \eta - \mathsf{increment})$ with Algorithm 4
  if success **then**
    set $\mathsf{found} = \mathsf{out}$
    set $\eta = \eta - \mathsf{increment}$
until $\mathsf{increment} \leq \mathsf{precision}$
output found

complexity bounded by 2^η. Once we have reached this vertex, we construct the chain by going backwards, following the Best pointers.

We easily prove what follows by induction.

Lemma 1. *At the end of the iteration of Algorithm 4 for (j, η_2, st'), $\Delta_{j,\eta_2}^{st'}$ is the maximum of $\log_2 \delta^2$, where δ is the bias obtained by an Round_L-rounded simple chain from a vertex of form $(k, \eta_1, 0)$ to (j, η_2, st') with max-complexity bounded by 2^η ($\Delta_{j,\eta_2}^{st'} = -\infty$ if there is no such chain).*

Lemma 2. *If there exists a simple Round_L-rounded chain c ending on state (k_j, η_j, st_j) and max-complexity bounded by 2^η, there exists one c' such that $\Delta_{i,\eta_i}^{st_i} = \log_2 \delta_i^2$ at each step.*

Proof. Let c'' be a simple chain ending on (k_j, η_j, st_j) with $\Delta_{j\eta_j}^{st_j} = \log_2 \delta_j^2$. Let $(k_{j-1}, \eta_{j-1}, st_{j-1})$ be the preceding vertex in c''. We apply Lemma 2 on this vertex by induction to obtain a chain c'''. Since the complexity of the last edge does not depend on the bias and $\alpha \geq 0$ in the last edge, we construct the chain c', by concatenating c''' with the last edge of c''. □

Theorem 2. *Algorithm 4 finds a θ-valid simple Round_L-rounded chain for $\mathsf{LPN}_{k,\tau}$ with max-complexity bounded by 2^η if there exists one.*

Proof. We use Lemma 2 and the fact that increasing δ^2 keeps constraint (1) valid. □

If we used $L = \mathbb{R}$, Algorithm 4 would always find a valid simple chain with bounded max-complexity when it exists. Instead, we use rounded chains and hope that rounding still makes us find the optimal chain.

Algorithm 4 Search for a best LPN reduction chain with max-complexity bounded to η

```
Input: k, τ, θ, L, η
Output: a valid simple rounded chain with max-complexity bounded to η

δ = 1 − 2τ
Construct the graph G using Algorithm 2 with parameters k, τ, L, η
for all η_1 ∈ L do
    set Δ^0_{k,η_1} = log_2 δ^2, Best^0_{k,η_1} = ⊥
    set Δ^{st}_{k,η_1} = −∞, Best^{st}_{k,η_1} = ⊥   ▷ Δ^{st} stores the best bias for a vertex (k, η_1, st) in a chain,
  and Best^{st} is the edge ending to this vertex in this chain
for j : k downto 1 do                                ▷ Search for the optimal chain
    for η_2 ∈ L in decreasing order do
        set Δ^{st}_{j,η_2} = 0, Best^{st} = ⊥ for all st
        foreach st' and each edge e to (j, η_2, st')
            set (i, η_1, st) to the origin of e and α and β as defined by e
            if αΔ^{st}_{i,η_1} + β ≥ Δ^{st'}_{j,η_2} then set Δ^{st'}_{j,η_2} = αΔ^{st}_{i,η_1} + β, Best^{st} = e
        if η_2 > 1 − Δ^{st'}_{j,η_2} + 2 log_2(−φ^{−1}(1 − (1 − θ)^{1/(2^j − 1)})) and j + log_2 j ≤ η then
            Construct the chain c ending by Best^{st'}_{j,η_2} and output (c, true)
output (⊥, false)
```

So, we build Algorithm 3. In this algorithm, we look for the minimal η for which Algorithm 4 returns something by a divide and conquer algorithm. First, we set η as being in the interval $[0,k]$ where the solution for $\eta = k$ corresponds to a brute-force search. Then, we cut the interval in two pieces and see if the lower interval has a solution. If it does, we iterate in this interval. Otherwise, we iterate in the other interval. We stop once the amplitude of the interval is lower than the requested precision. The complexity of Algorithm 3 is of $\log_2 \frac{k}{\mathsf{precision}}$ calls to Algorithm 4.

Theorem 3. *Algorithm 3 finds a* θ*-valid simple* Round_L*-rounded chain for* $\mathsf{LPN}_{k,\tau}$ *with parameter* $\mathsf{precision}$*, with optimal rounded max-complexity, where the rounding function approximates* $\log_2$ *up to* $\mathsf{precision}$ *if there exists one.*

Proof. Algorithm 3 is a divide-and-conquer algorithm to find the smallest η such that Algorithm 4 finds a valid simple Round_L-rounded chain of max-complexity bounded by 2^η. □

We can see that the complexity of Algorithm 4 is of $O\left(k^2 \cdot |L|\right)$ iterations as vertices have k possible values for the secret length and $|L|$ possible values for the logarithmic number of equations. So, it is linear in the size of the graph. Furthermore, each type of edge to a fixed vertex has $O(k)$ possible origins. The memory complexity is $O(k \cdot |L|)$, mainly to store the $\Delta_{k,\eta}$ and $\mathsf{Best}_{k,\eta}$ tables. We also use Algorithm 1 which has a complexity $O(k^3)$ but we run it only once during precomputation. Algorithm 3 sets $|L| \sim \frac{k}{\mathsf{precision}}$. So, the complexity of Algorithm 3 is $O\left(k^3 + \frac{k^3}{\mathsf{precision}} \times \log \frac{k}{\mathsf{precision}}\right)$.

6 Chains with a Guessing Step

In order to further improve our valid chain we introduce a new reduction step to our algorithm. As it is done in previous works [23,5], we guess part of the bits of the secret. More precisely, we assume that b bits of the secret have a Hamming weight smaller or equal to w. The influence on the whole algorithm is more complicated: it requires to iterate the WHT step $\sum_{i=0}^{w} \binom{w}{i}$ times. The overall complexity must further be divided by $\sum_{i=0}^{w} \binom{w}{i} \left(\frac{1-\delta_s}{2}\right)^{i} \left(\frac{1+\delta_s}{2}\right)^{w-i}$. Note that this generalized $\mathit{guess\text{-}secret}$ step was used in Guo et al. [23].

We formalize this step as following:

– $\mathit{guess\text{-}secret}(b,w)$ guesses that b bits of the secret have a Hamming weight smaller or equal to w. The b positions are chosen randomly. The number of queries remains the same, the noise is the same and the size of the secret is reduced by b bits. Thus, for this step we have

$\mathit{guess\text{-}secret}(b,w) : k' = k - b;\ n' = n;\ \delta' = \delta;\ \delta'_s = \delta$ Complexity: $O(nb)$ (included in $\mathit{sparse\text{-}secret}$) and the Walsh transform has to be iterated $\sum_{i=0}^{w} \binom{w}{i}$ times and the complexity of the whole algorithm is divided by $\sum_{i=0}^{w} \binom{w}{i} \left(\frac{1-\delta_s}{2}\right)^{i} \left(\frac{1+\delta_s}{2}\right)^{w-i}$

This step may be useful for a sparse secret, i.e. τ is small, as then we reduce the size of the secret with a very small cost. In order to accommodate this new step we would have to add a transition from state 3 to state 3 in the automaton that accepts the simple chains (See Figure 1).

To find the optimal chain using $\mathit{guess\text{-}secret}(b,w)$, we have to make a loop over all possible b and all possible w. We run the full search $O(k^2)$ times. The total complexity is thus $O\left(\frac{k^5}{\mathsf{precision}} \times \log \frac{k}{\mathsf{precision}}\right)$.

7 Results

We illustrate in this section the results obtained by running Algorithm 4 for different LPN instances taken from Bogos et al. [7]. They vary from taking

$k = 32$ to $k = 768$, with the noise levels: $0.05, 0.1, 0.125, 0.2$ and 0.25. In Table 3 we display the logarithmic time complexity we found for solving LPN without using *guess-secret*.[9]

τ	k						
	32	48	64	100	256	512	768
0.05	$13.89^{11.26}_{0.1}$	$14.52^{12.94}_{0.1c}$	$16.04^{14.43}_{0.1c}$	$20.47^{18.46}_{0.1c}$	$36.75^{34.45}_{0.1c}$	$57.77^{55.09}_{0.1c}$	$76.63^{74.03}_{0.1c}$
0.1	$15.04^{12.70}_{0.1}$	$18.58^{16.43}_{0.1}$	$21.58^{19.38}_{0.1c}$	$27.61^{25.39}_{0.1c}$	$46.75^{44.22}_{0.1c}$	$73.68^{70.92}_{0.1c}$	$98.97^{96.04}_{0.1c}$
0.125	$15.66^{13.52}_{0.1}$	$19.29^{17.00}_{0.1}$	$22.94^{20.50}_{0.1}$	$28.91^{26.30}_{0.1}$	$49.90^{47.35}_{0.1c}$	$78.85^{76.22}_{0.1c}$	$105.89^{103.01}_{0.1c}$
0.2	$17.01^{14.80}_{0.1}$	$21.25^{19.23}_{0.1}$	$24.42^{22.00}_{0.1}$	$32.06^{29.75}_{0.1}$	$56.31^{53.82}_{0.1c}$	$89.04^{86.38}_{0.1c}$	$121.04^{118.18}_{0.1c}$
0.25	$18.42^{16.30}_{0.1}$	$22.34^{20.43}_{0.1}$	$26.86^{24.58}_{0.1}$	$32.94^{30.75}_{0.1}$	$59.47^{56.88}_{0.1}$	$94.66^{91.97}_{0.1c}$	$127.35^{124.63}_{0.1c}$

entry of form $a^{b}_{c\ldots}$: $a = \log_2$ complexity, $b = \log_2$ max-complexity, $c =$ precision
subscript c means that a *code-reduce* is used

Table 3: Logarithmic time complexity on solving LPN without *guess-secret*

τ	k						
	32	48	64	100	256	512	768
0.05	$11.85^{10.90}_{0.1cg13o}$	$13.01^{12.52}_{0.1cg23o}$	$14.44^{13.74}_{0.1cg38o}$	$17.20^{16.19}_{0.1cg75o}$	$30.13^{28.02}_{0.1cg178o}$	$49.56^{47.29}_{1cg417o}$	$68.15^{65.98}_{1cg682o}$
0.1	$12.41^{11.65}_{0.1cg23o}$	$15.23^{14.25}_{0.1cg37o}$	$17.71^{16.76}_{0.1cg52o}$	$24.02^{22.14}_{0.1cg77o}$	$45.99^{43.49}_{0.1cg100o}$	$73.68^{71.09}_{1cg2}$	$99.21^{96.34}_{1cg5}$
0.125	$13.30^{12.40}_{0.1cg26o}$	$16.49^{15.49}_{0.1cg39o}$	$20.57^{18.61}_{0.1cg36o}$	$27.14^{24.80}_{0.1cg47o}$	$49.90^{47.35}_{0.1c}$	$78.97^{76.24}_{1cg1}$	$106.18^{103.42}_{1cg4}$
0.2	$17.01^{14.80}_{0.1o}$	$21.25^{19.23}_{0.1o}$	$24.42^{22.00}_{0.1}$	$32.06^{29.75}_{0.1}$	$56.34^{53.82}_{0.1cg1}$	$89.28^{86.79}_{1cg3}$	$121.12^{118.57}_{1}$
0.25	$18.42^{16.30}_{0.1}$	$22.34^{20.43}_{0.1}$	$26.86^{24.58}_{0.1}$	$32.94^{30.75}_{0.1}$	$59.47^{56.88}_{0.1}$	$94.85^{92.36}_{1cg2}$	$127.63^{125.01}_{1cg3}$

entry of form $a^{b}_{c\ldots}$: $a = \log_2$ complexity, $b = \log_2$ max-complexity, $c =$ precision
subscript c means that a *code-reduce* is used
subscript o means that a only 1 bit of the secret is found by WHT
subscript gb means that a *guess-secret*$(b, \cdot)$ is used

Table 4: Logarithmic time complexity on solving LPN with *guess-secret*

[9] Complete results are provided as an additional material to this paper.

Sequence of chains. If we analyze in more details one of the chains that we obtained, e.g. the chain for $\mathsf{LPN}_{512,0.125}$, we can see that it first uses a *sparse-secret*. Afterwards, the secret is reduced by applying 5 times the *xor-reduce* and one *code-reduce* at the end of the chain. With a total complexity of $2^{79.46}$ and $\theta < 33\%$ it recovers 64 bits of the secret.

$$(512,63.3) \xrightarrow{sparse-secret} (512,63.3) \xrightarrow{xor-reduce(59)} (453,66.6) \xrightarrow{xor-reduce(65)}$$
$$(388,67.2) \xrightarrow{xor-reduce(66)} (322,67.4) \xrightarrow{xor-reduce(66)} (256,67.8) \xrightarrow{xor-reduce(67)}$$
$$(189,67.6) \xrightarrow{code-reduce} (64,67.6) \xrightarrow{\mathsf{WHT}}$$

The code used is a $[189,64]$ concatenation made of ten random codes: one instance of a $[18,6]$ code, five instances of a $[19,6]$ code, and four instances of a $[19,7]$ code. By manually tuning the number of equations without rounding, we can obtain with $n = 2^{63.299}$ a complexity of $2^{78.84}$. This is the value from Table 1.

On the guess-secret reduction. Our results show that the *guess-secret* step does not bring any significant improvement. If we compare Table 3 with Table 4 we can see that in few cases the guess step improves the total complexity. For $k \geq 512$, some results are not better than Table 3. This is most likely due to the lower precision used in Table 4.

We can see several cases where, at the end of a chain with *guess-secret*, only one bit of the secret is recovered by WHT. If only 1 bit of the secret is recovered by non-bruteforce methods, the next chain for $\mathsf{LPN}_{k-1,\tau}$ will have to be run several times, given the *guess-secret* step used in the chain for $\mathsf{LPN}_{k,\tau}$. Thus, it might happen that the first chain does not dominate the total complexity. So, our strategy to use sequences of chains has to be revised, but most likely, the final result will not be better than sequences of chains without *guess-secret*. So, we should rather avoid these chains ending with 1 bit recovery.

There is no case where a *guess-secret* without a chain ending with 1 bit brings any improvement.

Comparing the results. For practical values we compare our results with the previous work [23,29,37,7].

From the work of ASIACRYPT'14 [23] and EUROCRYPT'16 [37] we have that $\mathsf{LPN}_{512,0.125}$ can be solved in time complexity of $2^{79.9}$ (with more precise complexity estimates). The comparison was shown in Table 1 in Introduction. We do better, provide concrete codes and we even remove the *guess-secret* step with an optimized use of a code. Thus, the results of Algorithm 4 improve all the existing results on solving LPN.

8 Conclusion

In this article we have proposed an algorithm for creating reduction chains with the optimal max-complexity. The results we obtain bring improvements to the existing work and to our knowledge we have the best algorithm for solving $\mathsf{LPN}_{512,0.125}$. We believe that our algorithm could be further adapted and automatized if new reduction techniques would be introduced.

As future works, we could look at applications to the LWE problem. Kirchner and Fouque [28] improve the LWE solving algorithms by refining the modulus switching. We could also look at ways to keep track of biases of secret bits bitwise, in order to allow cascades of *code-reduce* steps.

References

1. Michael Alekhnovich. More on Average Case vs Approximation Complexity. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 298–307. IEEE Computer Society, 2003.
2. V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economical construction of the transitive closure of a directed graph. 1970.
3. Tsonka Stefanova Baicheva, Iliya Bouyukliev, Stefan M. Dodunekov, and Veerle Fack. Binary and ternary linear quasi-perfect codes with small dimensions. *IEEE Transactions on Information Theory*, 54(9):4335–4339, 2008.
4. Daniel J. Bernstein. Optimizing linear maps modulo 2. `http://binary.cr.yp.to/linearmod2-20090830.pdf`.
5. Daniel J. Bernstein and Tanja Lange. Never Trust a Bunny. In Jaap-Henk Hoepman and Ingrid Verbauwhede, editors, *Radio Frequency Identification. Security and Privacy Issues - 8th International Workshop, RFIDSec 2012, Nijmegen, The Netherlands, July 2-3, 2012, Revised Selected Papers*, volume 7739 of *Lecture Notes in Computer Science*, pages 137–148. Springer, 2012.
6. Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 435–440. ACM, 2000.
7. Sonia Bogos, Florian Tramèr, and Serge Vaudenay. On solving LPN using BKW and variants - Implementation and analysis. *Cryptography and Communications*, 8(3):331–369, 2016.
8. Sonia Bogos and Serge Vaudenay. How to sequentialize independent parallel attacks? - biased distributions have a phase transition. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 704–731. Springer, 2015.
9. Sonia Bogos and Serge Vaudenay. Observations on the LPN Solving Algorithm from Eurocrypt'16. Cryptology ePrint Archive, Report 2016/451, 2016. `https://eprint.iacr.org/2016/451`.
10. Julien Bringer, Hervé Chabanne, and Emmanuelle Dottax. HB^{++}: a Lightweight Authentication Protocol Secure against Some Attacks. In *Second International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing (SecPerU 2006), 29 June 2006, Lyon, France*, pages 28–33. IEEE Computer Society, 2006.

11. José Carrijo, Rafael Tonicelli, Hideki Imai, and Anderson C. A. Nascimento. A Novel Probabilistic Passive Attack on the Protocols HB and HB^+. *IEICE Transactions*, 92-A(2):658–662, 2009.
12. Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors. *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*. Springer, 2005.
13. G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein. *Covering Codes*. North-Holland Mathematical Library. Elsevier Science, 1997.
14. Gérard D. Cohen, Mark G. Karpovsky, H. F. Mattson Jr., and James R. Schatz. Covering radius - survey and recent results. *IEEE Transactions on Information Theory*, 31(3):328–343, 1985.
15. Ivan Damgård and Sunoo Park. Is Public-Key Encryption Based on LPN Practical? *IACR Cryptology ePrint Archive*, 2012:699, 2012.
16. Nico Döttling, Jörn Müller-Quade, and Anderson C. A. Nascimento. IND-CCA Secure Cryptography Based on a Variant of the LPN Problem. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 485–503. Springer, 2012.
17. Alexandre Duc and Serge Vaudenay. HELEN: A Public-Key Cryptosystem Based on the LPN and the Decisional Minimal Distance Problems. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings*, volume 7918 of *Lecture Notes in Computer Science*, pages 107–126. Springer, 2013.
18. Tuvi Etzion and Beniamin Mounits. Quasi-perfect codes with small distance. *IEEE Transactions on Information Theory*, 51(11):3938–3946, 2005.
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 Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2006.
20. Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. $HB^\#$: Increasing the Security and Efficiency of HB^+. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 361–378. Springer, 2008.
21. Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. How to Encrypt with the LPN Problem. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 679–690. Springer, 2008.
22. Ronald L. Graham and Neil J. A. Sloane. On the covering radius of codes. *IEEE Transactions on Information Theory*, 31(3):385–401, 1985.
23. Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN Using Covering Codes. In Palash Sarkar and Tetsu Iwata, editors, *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, volume 8873 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2014.

24. Nicholas J. Hopper and Manuel Blum. Secure Human Identification Protocols. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001.
25. Ari Juels and Stephen A. Weis. Authenticating Pervasive Devices with Human Protocols. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.
26. Eike Kiltz, Daniel Masny, and Krzysztof Pietrzak. Simple Chosen-Ciphertext Security from Low-Noise LPN. In Hugo Krawczyk, editor, *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2014.
27. Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient Authentication from Hard Learning Problems. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 7–26. Springer, 2011.
28. Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 43–62. Springer, 2015.
29. Éric Levieil and Pierre-Alain Fouque. An Improved LPN Algorithm. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2006.
30. Vadim Lyubashevsky. The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem. In Chekuri et al. [12], pages 378–389.
31. Vadim Lyubashevsky and Daniel Masny. Man-in-the-Middle Secure Authentication Schemes from LPN and Weak PRFs. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 308–325. Springer, 2013.
32. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding Random Linear Codes in O(2ˆ{0.054n}). In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011.
33. W.W. Peterson and E.J. Weldon. *Error-correcting Codes*. MIT Press, 1972.
34. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005.
35. Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *J. Cryptology*, 21(1):131–147, 2008.

36. Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.
37. Bin Zhang, Lin Jiao, and Mingsheng Wang. Faster algorithms for solving LPN. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 168–195. Springer, 2016.