The Learning Parity with Noise Problem

Sonia Mihaela Bogos
LASEC, I&C, EPFL

Abstract—The Learning Parity with Noise problem (LPN) is a well-known hard problem studied in cryptography and coding theory. It is believed that the LPN problem is resistant to quantum computers and it is an alternative to other number-theoretic problems (e.g., factorization and discrete logarithm) that can be solved easily on quantum computers. Also, due to its simplicity, it is a good candidate for lightweight devices.

In this write-up we study the LPN problem together with its applications in cryptography. We focus also on LPN solving algorithms. Our goal is to see what are the lower bounds for solving the problem and further, by using proper parameters, to construct a fully-homomorphic encryption scheme based on LPN.

Index Terms—Learning Parity with Noise, Public key encryption, Gaussian elimination, Bernoulli distribution.

I. INTRODUCTION

Before presenting the LPN problem together with its applications and the algorithms solving it, we introduce here the notations used and the security notions needed for the understanding of the following sections.

A. Preliminaries

Let \( \langle \cdot, \cdot \rangle \) denote the inner product, \( \mathbb{Z}_2 = \{0, 1\} \), and let \( \oplus \) denote the bitwise XOR. We denote by \( \lceil n \rceil \) the nearest integer of the real number \( n \). For \( n = u + \frac{1}{2} \), where \( u \in \mathbb{Z} \), \( \lceil n \rceil = u \). We denote the Hamming weight of a vector \( v \) by \( HW(v) \).

For a distribution \( D \), we denote by \( x \leftarrow D \) the fact that \( x \) is sampled according to \( D \). For a domain \( \mathcal{F} \), we denote by \( x \leftarrow \mathcal{F} \) the fact that \( x \) is drawn uniformly at random from \( \mathcal{F} \).

Let \( R \) be \( \mathbb{Z}_2[x]/f \), the ring of polynomials with coefficients in \( \mathbb{Z}_2 \) modulo some polynomial \( f \) of degree \( k \). A vector \( o \) of size \( k \) is represented as \( o = (a_1, \ldots, a_k) \), i.e., we start with index 1. We represent a polynomial \( f = \sum_{i=0}^{k-1} f_i x^i \) as a vector \( f = (f_1, f_2, \ldots, f_k) \) of size \( k \). Let \( \tau \in [0, \frac{1}{2}] \) be a noise parameter. By \( \text{Ber}_\tau \), we denote the Bernoulli distribution with parameter \( \tau \) (i.e., \( \Pr[\text{Ber}_\tau[1] = 1] = \tau \)). \( \text{Ber}_\tau^k \) represents the distribution over vectors \( v \) of size \( k \), where each component is sampled independently according to \( \text{Ber}_\tau \). The binary entropy function for \( \epsilon \in [\text{Ber}_\tau] \) is defined as \( H_2(\epsilon) = -\tau \log \tau - (1 - \tau) \log (1 - \tau) \).

A function \( f : \mathbb{N} \rightarrow \mathbb{R} \) is negligible if for every \( \epsilon \in \mathbb{N} \), there is \( n_\epsilon \in \mathbb{N} \) such that \( f(x) < \epsilon^{-x} \), for all \( x \geq n_\epsilon \).

An affine transformation \( \phi : \mathbb{Z}_2^k \rightarrow \mathbb{Z}_2^k \) is defined by a matrix \( X \in \mathbb{Z}_2^{k \times k} \) and a vector \( x \in \mathbb{Z}_2^k \) as \( \phi(x) = Xr + x \).

The Hadamard matrix can be defined recursively as: \( H_0 = 1 \), \( H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{l-1} \quad H_{l-1} \end{pmatrix} \), for any \( l > 0 \). The Walsh-Hadamard transform (WHT) can be defined as \( \hat{x} = H_l x \), where \( x \) and \( \hat{x} \) are the signal and spectrum vectors of size \( 2^l \). The complexity of the WHT is \( O(2^l) \). The Fast Walsh-Hadamard transform is a divide and conquer algorithm that computes the WHT in \( O(l2^l) \).

B. Security Notions

Definition 1 (Public key encryption) A public key encryption scheme \( PKE = (\text{KeyGen}, \text{Enc}, \text{Dec}) \) consists of three polynomial-time algorithms:

- \( \text{KeyGen}(1^k) \) is a probabilistic algorithm. It takes as input the security parameter \( z \) and generates a secret key and a public key: \( (sk, pk) \).
- \( \text{Enc}_{pk}(m) \) is a probabilistic algorithm that takes as input the public key \( pk \) and a message \( m \in \mathcal{M} \) from the message space \( \mathcal{M} \) and produces the ciphertext \( c \) in the ciphertext space \( \mathcal{C} \).
- \( \text{Dec}_{sk}(c) \) is a deterministic algorithm taking as input the secret key \( sk \) and the ciphertext \( c \in \mathcal{C} \) and produces the plaintext \( m \) or an error.

We require for the PKE = (KeyGen, Enc, Dec) to be correct; i.e., for all messages \( m \in \mathcal{M} \) we have \( \Pr[\text{Dec}_{sk}(\text{Enc}_{pk}(m)) \neq m \mid (pk, sk) \leftarrow \text{KeyGen}(1^k)] \leq \text{negl}(z) \), where \( \text{negl}(z) \) is a negligible function.
We will use the following three security notions: IND-CPA (Indistinguishable under Chosen Plaintext Attack), IND-CCA1 (Indistinguishable under Non-adaptive Chosen Ciphertext Attack), and IND-CCA2 (Indistinguishable under Adaptive Chosen Ciphertext Attack). First, we define an adversary \( \mathcal{A} \) as a pair of probabilistic polynomial algorithms \( \mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \). The oracles to which \( \mathcal{A} \) has access are denoted by \( O_i \), with \( i \in \{1, 2\} \). We define \( O_i(\cdot) = \perp \), \( i \in \{1, 2\} \), to say that the oracle \( O_i \) returns the empty string \( \perp \) for any input.

In the scenario of IND-CPA, IND-CCA1, and IND-CCA2, the adversary has to distinguish between the encryption of two messages, \( m_0 \) and \( m_1 \), chosen by him. In the IND-CPA case, \( \mathcal{A} \) receives as input the public key \( pk \). It chooses \( m_0 \) and \( m_1 \), two messages in \( \mathcal{M} \). As a challenge, he is given \( c^* \) \( \leftarrow \text{Enc}_{sk}(m_b) \) with \( b \leftarrow \{0, 1\} \) and he has to say what plaintext was encrypted. In the IND-CCA1 scenario, the adversary is given access to a decryption oracle before choosing \( m_0 \) and \( m_1 \). The IND-CCA2 adversary has access to the decryption oracle even after he is given \( c^* \). The only restriction is that he is not allowed to submit a decryption request for \( c^* \). Informally, a scheme PKE is IND-\{CPA, CCA1, CCA2\} secure if there is no better strategy for the adversary than to guess randomly which of the two messages was encrypted. More formally, we define the three security notions below.

**Definition 2** For \( X \in \{\text{CPA, CCA1, CCA2}\} \), we say that the scheme PKE = (\text{KeyGen, Enc, Dec}) is IND-\( X \)-secure if it holds that for every adversary \( \mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \)

\[
\Pr[A_{\mathcal{O}^2}(r) = b | (m_0, m_1, s) \leftarrow \text{KeyGen}(1^k), b \leftarrow U(0, 1), c^* \leftarrow \text{Enc}_{sk}(m_b)] - \frac{1}{2} \leq \text{negl}(z),
\]

where

- \( r = (m_0, m_1, s, c^*) \)
- for \( X = \text{CPA} \), then \( O_1(\cdot) = \perp \) and \( O_2(\cdot) = \perp \)
- for \( X = \text{CCA1} \), then \( O_1(\cdot) = \text{Dec}_{pk}(\cdot) \) and \( O_2(\cdot) = \perp \)
- for \( X = \text{CCA2} \), then \( O_1(\cdot) = \text{Dec}_{pk}(\cdot) \) and \( O_2(\cdot) = \text{Dec}_{sk}(\cdot) \)


**C. Cryptographic primitives**

**Definition 3** (One-time signature) A one-time signature \( \text{SIG} = \langle \text{Gen, Sign, Verify} \rangle \) consists of three probabilistic polynomial-time algorithms:

- \( \text{Gen}(1^k) \) takes as input the security parameter \( k \) and outputs a pair of verification and signature keys: \( (vk, skg) \).
- \( \text{Sign}_{skg}(m) \) takes a message from the message space \( m \in \mathcal{M} \) and a signature key \( skg \) and produces a signature \( \sigma \).
- \( \text{Verify}_{vk}(m, \sigma) \) takes as input the verification key \( vk \), the message \( m \) and the signature \( \sigma \) and outputs a bit \( b \).

We require for the \( \text{SIG} = \langle \text{Gen, Sign, Verify} \rangle \) to be correct; for all messages \( m \in \mathcal{M} \) holds that

\[
\Pr[\text{Verify}_{vk}(m, \text{Sign}_{skg}(m)) = 1 | (vk, skg) \leftarrow \text{Gen}(1^k)] \geq 1 - \text{negl}(z),
\]

where \( \text{negl}(z) \) is a negligible function. For the one-time signature we require Existential Unforgeability under Chosen Message Attacks (EUF-CMA); i.e. for every adversary \( \mathcal{A} \) the probability of being able to construct a valid signature \( \sigma \), when given access once to a signing oracle, is negligible.

II. THE LEARNING PARITY WITH NOISE PROBLEM (LPN)

THIS section introduces the LPN problem together with its variants and applications in cryptography.

**A. Defining LPN**

Intuitively, the LPN problem asks to recover a secret vector \( s \) given access to noisy inner products of itself and random vectors. Before defining the LPN problem, we introduce the notion of LPN oracle.

**Definition 4** (LPN oracle) Let \( s \leftarrow U(\mathbb{Z}_2^n) \), let \( \tau \in [0, \frac{1}{2}] \) be a constant noise parameter and let \( \text{Ber}_{\tau} \) be the Bernoulli distribution with parameter \( \tau \). Denote by \( A_{s,\tau} \) the distribution defined as

\[
\{(v, b) \mid v \leftarrow \text{Ber}_{\tau}, b = \langle v, s \rangle \oplus \epsilon, \epsilon \leftarrow \text{Ber}_{\tau} \in \mathbb{Z}_2^{k+1} \}.
\]

An LPN oracle \( A_{s,\tau} \) is an oracle which outputs independent random samples according to the \( A_{s,\tau} \) distribution.

**Definition 5** (Search LPN problem) Given access to an LPN oracle \( A_{s,\tau}^{\text{LPN}} \), find the vector \( s \).

We will use the notation \( \text{LPN}_{k,\tau} \) to say that we have an LPN instance where the secret has size \( k \) and the noise parameter is \( \tau \).

An equivalent way to formulate the search LPN\( k,\tau \) problem is as follows: given access to a random matrix \( A \in \mathbb{Z}_2^{n \times k} \) and a random column vector \( b \) over \( \mathbb{Z}_2^k \), such that \( As \oplus b = f \), 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 of \( s \) and the corresponding \( v \) vectors. The column vector \( d \) is of size \( n \) and contains the corresponding noise bits \( \epsilon \) for each query. Value \( n \) is the number of queries.

One may observe that with \( \tau = 0 \), the problem can be solved easily in polynomial time through Gaussian elimination given \( n = \Theta(k) \) queries. The LPN\( k,\tau \) problem becomes hard once noise is added to the inner product. Usually the value of \( \tau \) is constant and independent from the value of \( k \). A case where \( \tau \) is taken as a function of \( k \) is in the construction of the Alekhnovich’s cryptosystem [12], details of which will be given later. Intuitively, a larger value of \( \tau \) means more noise and makes the problem of LPN\( k,\tau \) harder. The value of the noise parameter is a trade-off between the hardness of the LPN\( k,\tau \) and the practical impact in the applications that rely on this problem.

We say that an algorithm \( \mathcal{M} \) \( (q, t, m, \theta) \)-solves the search LPN\( k,\tau \) problem if

\[
\Pr[\mathcal{M}_{A_{k,\tau}}^{\text{LPN}}(1^k) = s | s \leftarrow U(\mathbb{Z}_2^k)] \geq \theta
\]
and \( \mathcal{M} \) runs in time \( t \), uses memory \( m \) and asks at most \( q \) queries from the LPN oracle.

The decisional \( \text{LPN}_{k, \tau} \) problem is to distinguish between the uniform distribution over \( \mathbb{Z}_2^{k+1} \) and the samples given by an LPN oracle.

The following result shows that decisional and search \( \text{LPN}_{k, \tau} \) are equivalent:

**Lemma 1 ([19])** If there is an algorithm \( \mathcal{M} \) that \( (q, t, m, \theta) \)-solves the decisional \( \text{LPN}_{k, \tau} \), then one can build an algorithm \( \mathcal{M}' \) that \( (q', t', m', \theta') \)-solves the search \( \text{LPN}_{k, \tau} \) problem, where \( q' = O(q \cdot \theta^{-2} \log k), \ t' = O(t \cdot k \cdot \theta^{-2} \log k), \ m' = O(m \cdot \theta^{-2} \log k) \) and \( \theta' = \frac{q}{2} \).

The search \( \text{LPN} \) problem can be formulated also as the following optimization problem: given a random matrix \( A \) and a random column vector \( b \) over \( \mathbb{Z}_2 \), find the vector \( s \) that maximizes the number of equations of the system \( As = b \). This corresponds to the problem of decoding a random linear code and it was shown to be NP-hard [4]. Thus, \( \text{LPN}_{k, \tau} \) is a hard problem: there is no algorithm \( \text{LPN} \) for \( (q, t, m, \theta) \)-solves the problem, where \( q \) is polynomial in \( k \) and \( \frac{1}{2} \) and \( \theta \) is not a negligible function. So far the best algorithms to solve search \( \text{LPN} \) require sub-exponential number of queries and work in sub-exponential time.

**B. Variants of \( \text{LPN} \)**

There exists variants of the \( \text{LPN} \) problem, depending on how we choose the distribution for the secret \( s \) and noise parameter.

- **a) Non-Uniform Secrets:** In the definition of \( \text{LPN}_{k, \tau} \), \( s \) is an uniformly random vector of size \( k \). Assume we have a variant of \( \text{LPN}_{k, \tau} \) where the vector \( s \) is chosen from any distribution \( \mathcal{D} \) over \( \mathbb{Z}_2^k \). We denote this new problem by \( \text{LPN}_{k, \tau}^\mathcal{D} \). The classical version of \( \text{LPN}_{k, \tau} \) is at least as hard as \( \text{LPN}_{k, \tau}^\mathcal{D} \): an algorithm \( \mathcal{M} \) that solves \( \text{LPN}_{k, \tau} \) and recovers \( s \), can solve \( \text{LPN}_{k, \tau}^\mathcal{D} \) for any distribution \( \mathcal{D} \).

- **b) LPSON:** In [3], Arora and Ge introduced a new variant of the \( \text{LPN} \) problem where the noise bits can be characterized as the solution of a polynomial \( P \). We call this new variant Learning Parity with Structured Noise Problem (LPSON).

Instead of receiving queries of the form \( \langle v, (v, s) \oplus \epsilon \rangle \) from the LPN oracle, one receives a chunk of queries: \( \langle v_1, (v_1, s) \oplus \epsilon_1 \rangle, \ldots, \langle v_m, (v_m, s) \oplus \epsilon_m \rangle \), such that the noise bits respect \( P(\epsilon_1, \ldots, \epsilon_m) = 0 \). One example of such polynomial is \( P(\epsilon_1, \ldots, \epsilon_m) = \prod_{i=1}^m \epsilon_i \) that expresses that from the \( m \) queries received, at least one has the bit noise set to 0. The noise bits may still have the Bernoulli distribution \( \text{Ber}_p \) but they are not mutually independent any more. Being able to structure the noise gives extra information and it was proven that LPSON can be solved in time \( O(kf^m) \) [3], where \( f \) is the degree of the multilinear polynomial \( P \) in \( m \) variables. This is polynomial if \( m \) is constant.

- **c) Ring-LPN:** An extension of the LPN problem where we work in a ring, Ring-LPN, was introduced in [16].

**Definition 6** Let \( R \) be \( \mathbb{Z}_2[x]/f \), where \( f \) is a polynomial of degree \( k \). Let \( s \leftarrow U \) and let \( \tau \in [0, \frac{1}{2}] \) be a constant noise parameter. Denote by \( A_{R,s,\tau} \) the distribution defined as

\[
\{(v, b) \mid v \leftarrow U, b = vs + d, d \leftarrow \text{Ber}^k \} \in R \times R.
\]

A Ring-LPN oracle \( A_{R,s,\tau} \) is an oracle which outputs independent random samples according to the \( A_{R,s,\tau} \) distribution.

**Definition 7** (Search Ring-LPN \( R_{k, \tau} \) problem) Given access to an Ring-LPN oracle \( A_{R,s,\tau} \), find the polynomial \( s \).

Similar to decisional LPN, decisional Ring-LPN \( R_{k, \tau} \) asks to distinguish between the samples from the Ring-LPN oracle and \( (f, f') \) for random \( f, f' \in R \).

The advantage in Ring-LPN is that the product of two vectors, seen as two elements of the ring, gives another vector. Thus, we can replace the matrix \( A \) from the classical LPN by a ring element. An attack against the security claims of Ring-LPN was presented in [5]; i.e. the authors of [5] introduced an algorithm that solves the search Ring-LPN with less requirements than the ones claimed by [16].

One can forget the ring structure and transform a Ring-LPN instance where we are given \( n \) queries into an LPN instance where we are given \( nk \) queries.

- **d) Subspace LPN:** Another variant of LPN, Subspace LPN, offers a degree of interactivity between the algorithm \( \mathcal{M} \) that solves LPN and the LPN oracle. \( \mathcal{M} \) does not get queries of the form \( \langle v, (v, s) \oplus \epsilon \rangle \), but can submit to the oracle two affine transformations \( \phi_0 \) and \( \phi_1 \) and get queries of the form \( \langle v, (\phi_0(v), \phi_1(s)) \oplus \epsilon \rangle \). Given that the two affine transformations overlap in a \( k' \) dimensional subspace, Subspace LPN \( R_{k, \tau} \) is equivalent to LPN \( k', \tau \) [25].

- **e) Exact LPN:** We are given an LPN \( R_{k, \tau} \) instance, i.e. we are given \( (A, As \oplus d) \), where \( s \) is a random vector of size \( k \), \( A \) is a binary random matrix of size \( n \times k \) and \( d \) is the noise vector where each bit is drawn from distribution \( \text{Ber}_p \). A variant of the LPN \( R_{k, \tau} \), called XLPN \( R_{k, \tau} \), is defined as LPN \( k, \tau \) where the Hamming weight of \( d \) is exactly \( \lceil n \tau \rceil \).

Given that search LPN \( k, \tau \) is hard, search XLPN \( k, \tau \) is hard. It is an open problem to say whether decisional XLPN \( k, \tau \) is equivalent to decisional LPN \( k, \tau \) [24].

- **f) Sparse LPN:** One scenario that may create an\( LPN_{k, \tau} \) instance easy to solve is when the secret vector \( s \) is sparse. In this case, one may think to run exhaustive search in order to find \( s \). In [21] it was introduced a transformation which takes an LPN \( k, \tau \) instance \( (A, As \oplus d) \) with \( s \) as a uniform random vector and gives an LPN \( \bar{R}_{k, \tau} \) instance \( (A', A'd \oplus d') \) where the secret value, \( d \), is now a vector following the same distribution as the noise. Below there are given details of how we can obtain this transformation.

Given the queries \( \langle v_1, b_1 \rangle, \ldots, \langle v_n, b_n \rangle \) from the LPN oracle, choose from the \( n \) vectors \( k \) linearly independent ones,
THE search LPN_{k,τ} problem is a hard problem. The best algorithms to solve LPN_{k,τ} run in sub-exponential time. More exactly, a secret of k bits can be recovered with $2^{O(\frac{k}{\tau^c})}$ queries in $2^{O(\frac{k}{\tau^c})}$ time. The first solving algorithm introduced in the literature was BKW [7]. There exists improvements of this algorithm [22], [14], [5] but they still remain in the same sub-exponential complexity. The algorithm from [5] uses the fact that the secret may be sparse. Lyubashevsky [23] gave a variant of the BKW algorithm that solves the LPN_{k,τ} problem in $2^{O(\frac{k}{\tau^c} \log k)}$ time but for a polynomial number of queries $k^{1+γ}$. Here, the LPN_{k,τ} problem with $k^{1+γ}$ queries was turned into a LPN(k, $τ'$) problem that uses BKW as a black-box, with $τ' = \frac{1}{2} - \frac{1}{2} \left( \frac{1-δ}{4} \right) \frac{1}{\tau^c k^c}$.

A. BKW

This subsection presents the BKW algorithm. First, we assume that k can be written as $k = ab$, where $a$ and $b$ are positive natural numbers. We define $δ = 1 - 2τ$.

The intuitive idea behind BKW is the following: assume that from the LPN oracle we are given a query of the form $(v, (v_i, s) \oplus e_i)$, where $HW(v) = 1$ and the 1 is on position $i$, with $1 \leq i \leq k$. Denote this vector as $e_i$. Then, $(e_i, s) \oplus s$ and the query can be rewritten $(e_i, s_i \oplus e_i)$. We have that $Pr[e = 1] = \frac{1-δ}{2} < \frac{1}{2}$. Thus, if we have more queries of the form $(e_i, s_i \oplus e_i)$, the majority of them will have the noise bit set on 0. For the majority of them, the last bit of the query will give the value of $s_i$. The value of $δ$ dictates how many queries we need in order to apply the majority rule: the smaller $δ$ is, the more queries we need.

If we can obtain queries of this form, $v = e_i$, for all possible values of $i$, then the complete value of $s$ can be recovered.

We now describe the steps performed by BKW and give its complexity.

The input given to the BKW algorithm is represented by $n$ queries: $(v_1, (v_1, s) \oplus e_1), \ldots, (v_n, (v_n, s) \oplus e_n)$, where $s$ is the secret random vector of size $k$, $v_i$ are random vectors of size $k$ and $e_i$ are noise bits sampled from $Ber_{1-δ}$ for $1 \leq i \leq n$.

First, we split the $k$ bit vectors $v_i$ into $a$ blocks of $b$ bits. As aforementioned, the goal of the algorithm is to obtain multiple queries of the form $(e_i, (e_i, s) \oplus e_i)$ for every possible value of $i$. We want to find a small number $l$ of vectors $v_{i_1}, \ldots, v_{i_l}$ from the original queries such that $v_{i_1} \oplus \cdots \oplus v_{i_l} = e_i$. The following lemma shows what is the bias of $e_i$ for $v_{i_1} \oplus \cdots \oplus v_{i_l} = e_i$.

**Lemma 2 (Lemma 3 from [7])** Let $(v_1, b_1), \ldots, (v_1, b_t)$ be the queries from $A^{\text{LPN}_{k,τ}}$. Then we have that

$$Pr[(v_1 \oplus \cdots \oplus v_1, s) = b_1 \oplus \cdots \oplus b_t] = \frac{1 + δ^t}{2}.$$
Definition 8 (Definition 2 from [22]) Let \( A_{s,δ,i} \) be the distribution defined as
\[
\{(v, (v, s) \oplus ε) \mid v \leftarrow \{0,1\}^{n-i}b \times \{0\}^i, ε \leftarrow \text{Ber}_{\frac{1}{2}}\}
\]

Let \( A_{\text{BKW}}^{s,δ,i} \) denote an oracle which outputs independent samples according to this distribution. We define the \((s,δ,i)\)-set of size \( n \) as \( n \) queries from the \( A_{\text{BKW}}^{s,δ,i} \) oracle.

Lemma 3 (Lemma 2 from [22]) Assume we are given an \((s,δ,i)\)-set of size \( n \). We can in time \( O(n) \) construct an \((s,δ^2,i+1)\)-set of size \( n - 2^b \).

Proof: An \((s,δ,i)\)-set represents a set of vectors where the last \( i \) blocks of size \( a \) are zero. Put these vectors in equivalence classes according to their value of their \((a-i)\)th block (the last non-zero block). As the size of a block is \( b \), we will have at most \( 2^b \) equivalence classes. For each class, choose at random a representative vector \( v \) and xor it with the rest of the vectors. As two vectors from the same class have the same \((a-i)\)th block, the xor of them will result in a vector that has the last \( i+1 \) blocks zero. Discard the vector \( v \). The new vectors represent the xor of two vectors from the \((s,δ,i)\)-set. From Lemma 2 we obtain that the new bias is \( δ^2 \). As \( v \) was chosen at random, these new vectors correspond to the \( A_{s,δ^2,i+1} \) distribution. We have discarded at most \( 2^b \) vectors, so in the end we are left with \( n - 2^b \) vectors.

The initial queries from the LPN oracle can be seen as an \((s,δ,0)\)-set of size \( n \). By applying Lemma 3 \((a-1)\) times we obtain a \((s,δ^{2^{a-1}},a-1)\)-set of size \( n - (a-1)2^b \). All these new queries are of the form \((v, (v, s) \oplus ε) \) where only the first block of \( v \) has non-zero values and \( ε \leftarrow \text{Ber}_{\frac{1}{2}} \).

If the size of the \((s,δ^{2^{a-1}},a-1)\)-set is at least \( 2^b \), then there is a probability of \( 1 - \frac{1}{2} \) that the query \((e_i, (e_i, s) \oplus ε), 1 \leq i \leq b \) appears in the set [7]. Here, the value \( ε \) is the base of the natural logarithm. If the query is not in the set, the algorithm repeats the above process with a new \((s,δ,0)\)-set of queries. The expected number of repetitions is constant.

From this \((s,δ^{2^{a-1}},a-1)\)-set, the algorithm keeps only the above mentioned queries \((e_i, (e_i, s) \oplus ε) = (e_i, s_i \oplus ε), 1 \leq i \leq b \). The following result says how many queries are needed in order to be able to recover the value of \( s_i \).

Lemma 4 (Lemma 3 from [22]) Let \( A_{s,δ,a}^{BKW} \) be the distribution defined as
\[
\{s_i \oplus ε \mid ε \leftarrow \text{Ber}_{\frac{1}{2}}\}
\]

Also, let \( A_{s,δ,a}^{BKW} \) denote an oracle which outputs independent samples according to this distribution. Then, it is possible to guess the value of \( s_i \) with \( δ \cdot 2^{-a} \) calls to the oracle with error probability bounded by \( 2^{-\frac{a}{2}} \).

Proof: We define a sample as a pair \((v_i, b)\), where \( v_i \) is a random bit and \( b = (v_i, s_i) \oplus ε \). We say that \((v_i, b)\) is compatible with \( s_i \) if \((v_i, s_i) = b \). We count to see the number of samples compatible with \( s_i \) for \( s_i = 1 \) and \( s_i = 0 \). We have \( \Pr[(v_i, s_i) = b] = \frac{1+δ^{2^{a-1}}}{2} \). In a pool of \( N \) samples, we expect to have \( N \cdot \frac{1+δ^{2^{a-1}}}{2} \) such equalities. The probability \( \Pr[(v_i, 1 - s_i) = b] = \frac{1}{2} \) and the expected number of times \((v_i, 1 - s_i) = b \) is \( \frac{N}{2} \) for \( N \) samples.

An error in guessing the value of \( s_i \) occurs when actually we have that \( 1 - s_i \) matches more times with the samples from the \( A_{s,δ,a}^{BKW} \) oracle than \( s_i \). This scenario can be bounded using a Chernoff bound [11]. We compute the probability \( p_1 \) that \( s_i \) matches at most \( N' \) samples and the probability \( p_2 \) that \( (1 - s_i) \) matches at least \( N' \) samples. As the expected values for these two cases are \( N/2 \) and \( N(1 + δ^{2^{a-1}})/2 \), we can take
\[ N' = N(1 + δ^{2^{a-1}})/2, \text{ with } 0 < α < 1. \]

The probability of error is \( \Pr[\text{error}] ≤ p_1 + p_2 \). By choosing \( N = 8δ^{2^{a-1}} \) and \( α = 3 - \sqrt{6} \), we obtain that \( p_1 ≤ e^{-\frac{α}{2}} \) and \( p_2 ≤ e^{-\frac{α}{2}} \).

So far, we showed how to guess the values of the bits from the first block of \( s \). We can use the same original queries for guessing the bits of \( s \) for the rest of the blocks. For the \( j \)th block, with \( 1 ≤ j ≤ a \), we apply the same technique, with the difference that when we apply Lemma 3 we leave the \( j \)th block as being the only non-zero block from the \((s,δ^{2^{a-1}},a-1)\)-set. Repeating the above process for every block, will give the complete value of \( s \).

A summary of the BKW algorithm is as follows: the algorithm receives \( n \) queries: \((v_1, b_1), \ldots, (v_n, b_n)\) from the LPN oracle. These are represented as an \((s,δ,0)\)-set of size \( n \). The vectors \( v_i \) are seen as \( a \) blocks of \( b \) bits \((k = ab)\). For each \( i \)th block, use the \((s,δ,0)\)-set and apply Lemma 3 \((a-1)\) times to obtain a \((s,δ^{2^{a-1}},a-1)\)-set of size \( n - (a-1)2^b \). For each bit from the \( i \)th block, we need \( δ \cdot 2^{-a} \) queries in the \((s,δ^{2^{a-1}},a-1)\)-set to be able to guess the value of that bit. Apply Lemma 4 for each bit and output the value of \( s \).

The following theorem summarizes the complexity of the BKW algorithm.

Theorem 1 (Theorem 1 from [22]) For \( k = ab \), the BKW algorithm \( n = O(20 \cdot \ln (4k) \cdot 2^bδ^{-2a} + (a-1)2^b), t = O(kαn), m = O(kαn), θ = \frac{1}{2} \)-solves the LPN\(_{k,ε}\) problem.

From Theorem 1 one may notice that the total number of queries needed to recover the secret is polynomial in \( δ^{-2a} \) and \( 2^a \). A large bias, thus a small value for \( a \), could give a small number of queries. But \( k = ab \), where \( k \) is a parameter fixed at the beginning of the algorithm. Thus, a small value \( a \) will give a large value for \( b \).

For \( δ = 2^{1-k^4} \), with \( 0 < η < 1 \), \( a \) can take the value \( (1-η) \log k \) and \( b \) can be \( \frac{k^4}{(1-η) \log k} \) in order to obtain a complexity of \( 2^{O(\frac{1}{\log k})} \) [23].

The BKW algorithm can be seen as a variant of the Gaussian elimination. Compared to the normal process, where we are zeroing the equations unknown by unknown, the process from Lemma 3 is zeroing by blocks of size \( b \). We need \( 2^b \) equations to zero a block for all equations. The process continues for \((a-1)\) blocks. At the end, because of the noise, we need more than one equation for each bit of the secret \( s_i \), with \( 1 ≤ i ≤ k \), to find their value.

B. Improvement of BKW

An improvement to the BKW algorithm is presented in [22]. The new algorithm is a variant of BKW that recovers the value
of \( s \) block by block (compared with BKW where \( s \) is recovered bit by bit). The change comes in the last step, in the way the \( (s, \delta^{2^{s-1}}, a-1) \)-set is managed. In the BKW algorithm this set of queries was filtered such that only queries of the form \((e_1, s, \xi)\) were kept. Most of the queries were not used further.

The improved algorithm keeps all the queries and tries to guess the value of a block of \( s \). Assume we guess the value of the first block of \( s \), denoted by \( s^1 \). It counts, for each possible value \( y \) the block can take, how many equalities of the form \((v, y) = b\) we have, where \((v, b)\) are from the \( (s, \delta^{2^{s-1}}, a-1) \)-set. Assume that the set of queries has size \( N \). Given that we have \( 2^b \) possible values for the block, the complexity of counting would be \( 2^bN \). We can improve the complexity by the following observation: for two values \( v, y \) that differ in only one bit, \( y_c \neq y'_c \), we can do a single counting for both values. The counting for \( y \) is the sum of

- number of times \((v, y) = b\) and \( v_c = 0\)
- number of times \((v, y) = b\) and \( v_c = 1\).

The counting for \( y' \) can be done as the sum of

- number of times \((v, y) = b\) and \( v_c = 0\)
- number of times \((v, y) \neq b\) and \( v_c = 1\).

If we take pairs of values that differ in only 1 bit, we can simplify the counting to \( \frac{2^k}{2^b} \). This split can be done \( b \) times so that in the end the complexity of counting is \( N \) and all the \( 2^b \) possible values for \( p \) are seen as a fast Walsh-Hadamard transform of \( 2^b \) sums. The complexity of this transform is \( b2^b \). Thus, the time complexity of this step is \( O(N + b2^b) \).

After the counting, the value of \( p \) with the highest number of matches is chosen to be the value of \( s^1 \). A failure scenario appears when there is another value \( s^1 \neq s^1 \) that has a higher record after counting. Similar to the bit scenario we can bound the probability of failure using a Chernoff bound.

**Lemma 5** Let \( A_{s, \delta, a, b} \) be the distribution defined as

\[
\{ (v, (s, \xi)) \mid \epsilon \leftarrow Ber_{\frac{1}{2}}, v \leftarrow U \{(0)^{(s-1)b} \times Z_2^b \times \{(a-1)b\}} \}
\]

Also, let \( A_{s, \delta, a, b} \) denote an oracle which outputs independent samples according to this distribution. Then, it is possible to guess the value \( s^1 \) of the \( bth \) block of \( s \) with \( (8b + c)\delta^{2^{s-1}} \) calls to the oracle with error probability bounded by \( e^{-\tau} + 2^b e^{-\frac{(\mu+1)\delta^{2^{s-1}}}{2^b}} \).

The following theorem gives the complexity of the new algorithm.

**Theorem 2 (Theorem 2 from [22])** For \( k = ab \), there is an algorithm that \((n = O(8b + 200)\delta^{2^b} + (a-1)2^b), t = O(\log n), \theta = \frac{1}{2}\) solves the LPN_{k,\epsilon} problem.

IV. LPN BASED CRYPTO SYSTEM

A scheme into a IND-CCA2 secure one. We present here all three schemes.

A. Hard Problems

The cryptosystems presented in this section base their security on the following two problems:

**Definition 9** (LPN oracle) Let \( \tau = \Theta(k^{-\frac{1}{2}-\eta}) \), with \( \eta > 0 \) and let \( s \leftarrow Ber_{\frac{1}{2}} \). Denote by \( A_{s, k} \) the distribution defined as

\[
\{ (v, b) \mid v \leftarrow U \{Z_2^k \}, b = \langle v, s \rangle + \epsilon, \epsilon \leftarrow Ber_{\frac{1}{2}} \} \in \{0,1\}^k.
\]

An LPN oracle \( A_{s, k} \) is an oracle which outputs independent random samples according to the \( A_{s, k} \) distribution.

**Definition 10** (Decisional LPN problem) Given access to a linear number of queries \( m = O(k) \) one has to decide whether the queries are distributed according to the LPN oracle \( A_{s,k} \) or are chosen uniformly from \( \{0,1\}^k \).

The definition of decisional LPN problem is a variant of the problem introduced by Alekhnovich in [1] with the only difference that the secret \( s \) is not chosen uniformly at random, but it is distributed according to \( Ber_{\frac{1}{2}} \). Besides the distribution of the secret, there are two other differences between LPN1 and the LPN problem. First, in LPN1, the noise parameter is not constant, but it depends on the value of \( k \). The second difference is that now we are given access only to a linear number of queries and not to arbitrarily polynomial of them.

The second hard problem is the matrix version of LPN1.

**Definition 11** (LPN oracle) Let \( m = \Theta(k), \tau = \Theta(k^{-\frac{1}{2}-\eta}) \), with \( \eta > 0 \) and let \( S \leftarrow Ber_{\frac{m}{k}} \). Denote by \( A_{S, k} \) the distribution defined as

\[
\{ (v, (s, \xi)) \mid v \leftarrow U \{Z_2^k \}, S = \{v \oplus E, E \leftarrow Ber_{\frac{m}{k}} \} \in \{0,1\}^k \}
\]

An LPN oracle \( A_{S, k} \) is an oracle which outputs independent random samples according to the \( A_{S, k} \) distribution.

**Definition 12** (Decisional LPN problem) Given access to a linear number of queries \( n = \Theta(k) \) one has to decide whether the queries are distributed according to the LPN oracle \( A_{S, k} \) or are chosen uniformly from \( \{0,1\}^k \).

Before presenting the schemes, we introduce a list of parameters. Let \( z \) be the security parameter, \( \eta > 0 \), \( k \in \Theta(z^{2/(1-2\eta)}) \), \( l_1, l_2, l_3 \in \Theta(z^{2/(1-2\eta)}) \) and \( \tau \), the noise parameter, is \( \tau = \Theta(z^{2/(1-2\eta)}) \).

During the encryption, noise will be introduced. In order to cope with this, error-correcting codes are used. Let \( G \in Z_2^{12 \times k} \) be the generator matrix of a binary linear error-correcting code \( C \) such that \( \text{Decod}_{C} \) is an efficient decoding algorithm that can correct up to \( 2\alpha \) errors, where \( \alpha \) is a positive constant. At decryption, a binary error-correcting code \( D \subset Z_2^{12} \) will be used to recover the original message. The code \( D \) has a decoding algorithm \( \text{Decod}_{D} \) that can correct up to \( \lambda \) errors, with \( \lambda \) a positive constant.
**B. IND-CPA Secure Scheme PKE**

The three algorithms, KeyGen, Enc and Dec are presented in Algorithms 1-3.

The KeyGen algorithm produces the pair of \((pk, sk)\) where \(sk\) is the secret matrix \(T\). The public key \(pk = (A, B, C)\) is represented by two random matrices, \(A\) and \(C\), and \(B\) computed as in step 3.

**Algorithm 2 PKE1 Enc**

**Input:** Public key \((A, B, C)\), message \(m \in \mathbb{Z}_2^n\).
**Output:** Encryption \(c\).
1. \(t \leftarrow U_{\Sigma^k}\), \(i \leftarrow \text{Encode}_G(t)\)
2. \(B_i \leftarrow \text{Ber}_{\ell_2}^{l_1}t, \ X_i \leftarrow \text{Ber}_{\ell_2}^{l_2}t\)
3. compute \(B = G + Ta + X\)
4. set \(pk = (A, B, C), sk = T\)

Given a messages \(m\) and a public key \(pk\), one first encodes the message with the code \(D\). The ciphertext \(c = (c_1, c_2, c_3)\) will allow to recover first the value of \(s\) and further the value of \(m\) as described in Algorithm 3.

**Algorithm 3 PKE1 Dec**

**Input:** Secret key \(T\), encryption \(c = (c_1, c_2, c_3)\).
**Output:** Message \(m\).
1. compute \(y = c_2 - Tc_1\)
2. If \(s = \text{Decode}_D(y)\) succeeds
   then \(m = \text{Decode}_D(c_3 - Cs)\)
   else \(\text{output } ⊥\)

The PKE1 scheme is correct. The scenario where the decryption would fail is when one of the decoding algorithms fails. We take \(γ > 0\) such that \(γτ < \lambda\) and \(β > 0\) such that \(2β + γτ < α\). Given that \(HW(s) < γτk\), \(HW(e_1) < γτl_1\) and \(HW(e_2) < γτl_2\), with overwhelming probability in \(z\), then the value \(y\) does not have more than \(αl_2\) errors and we can decode \(s\). Also \(HW(e_3) < γτl_3\) and the decoding algorithm of \(D\) will recover successfully the message \(m\).

**Theorem 3 (Theorem 1 from [12])** Assume that LPN1 and LPN2 are hard. Then the scheme PKE1 is IND-CPA secure.

**C. IND-CCA1 Secure Scheme PKE2**

The new scheme is a tag-based encryption scheme. Each ciphertext \(c\) is assigned a random tag \(t\). This tag derives the public and the secret key. A \(E \subseteq \Sigma^2\) q-ary code over alphabet \(\Sigma\) with minimum distance \(l_2\delta\) and dimension \(k\) is used to encode the tags.

**Algorithm 4 PKE2 KeyGen**

**Input:** Security parameter \(z\), matrix \(G\).
**Output:** \((pk, sk)\).
1. \(A \leftarrow U_{\mathbb{Z}_2^{l_1 \times k}}, C \leftarrow U_{\mathbb{Z}_2^{l_3 \times k}}\)
2. for every \(j \in \Sigma\)
   \(T_j \leftarrow \text{Ber}_{\ell_2}^{l_1}t, X_j \leftarrow \text{Ber}_{\ell_2}^{l_2}t\)
3. compute \(B_j = G + Ta + X_j\)
4. set \(pk = (A, B, C), sk = (T_j)_{j \in \Sigma}\)

The key generation algorithm follows the same steps as for PKE1 only that now we sample random matrices \(B\) and \(T\) for each \(j \in \Sigma\).

**Algorithm 5 PKE2 Enc**

**Input:** Public key \((A, (B_j)_{j \in \Sigma}, C)\), message \(m \in \mathbb{Z}_2^n\).
**Output:** Encryption \(c\).
1. write each \(B_j\) as \(B_j = (b_{j,1}, \ldots, b_{j,l_2})^T\)
2. \(t \leftarrow U_{\Sigma^k}\), \(i \leftarrow \text{Encode}_G(t)\)
3. construct \(B_i = (b_{i,1}, \ldots, b_{i,l_2})^T\)
4. \(s \leftarrow \text{Ber}_{\ell_2}^{l_1}t, e_1 \leftarrow \text{Ber}_{\ell_2}^{l_1}t, e_2 \leftarrow \text{Ber}_{\ell_2}^{l_1}t, e_3 \leftarrow \text{Ber}_{\ell_2}^{l_1}\)
5. compute \(c_1 = As + e_1, c_2 = Bs + e_2, c_3 = Cs + e_3 + \text{Encode}_D(m)\)
6. set \(c = (t, c_1, c_2, c_3)\)

Given a tag \(t\), we can derive the public and private key. The encoding of \(t\) will tell which rows from each matrix \(B_j\) will be used to construct the public key. The same is done for the secret key \((T_j)_{j \in \Sigma}\).

**Algorithm 6 PKE2 Dec**

**Input:** Secret key \((T_j)_{j \in \Sigma}\), encryption \(c = (t, c_1, c_2, c_3)\).
**Output:** Message \(m\).
1. write each \(T_j\) as \(T_j = (t_{j,1}, \ldots, t_{j,l_2})^T\)
2. compute \(t \leftarrow \text{Encode}_G(t)\)
3. \(T_i = (t_{i,1}, \ldots, t_{i,l_2})^T\)
4. compute \(y = c_2 - T_i c_1\)
5. If \(s = \text{Decode}_D(y)\) succeeds
   then compute \(m = \text{Decode}_D(c_3 - Cs)\)
   verify that \(HW(s) < γτk, HW(e_1) < γτl_1, HW(e_2) < γτl_2\) and \(HW(e_3) < γτl_3\)
   else \(\text{output } ⊥\)

**Theorem 4 (Theorem 2 from [12])** The scheme PKE2 is IND-CCA1 secure, provided that scheme PKE1 is IND-CPA and the parameters \(α, β, γ, q\) and \(δ\) suffice \(δ < 1 - \frac{1}{q} + 2β + γτ + 1 - δ < α\).

**D. IND-CCA2 Secure Scheme PKE3**

The scheme PKE2 is transformed into a IND-CCA2 scheme by using a one-time signature SIG. The signature generates a pair of verification and signing keys: \((vk, sk)\). In this new scheme the tag is not chosen uniformly from \(\Sigma^k\), but it is assigned to be the verification key \(vk\).

The key generation algorithm for PKE3 is the same as the one for PKE2 (See Algorithm 4).
The ciphertext is not chosen uniformly at random but the encryption algorithm of PKE. 

The decryption algorithm of PKE first verifies that the signature is valid. That is, given \(c = (c', \sigma)\), it checks if SIG.Verify \(c, \sigma\). If the signature is valid, it runs the decryption algorithm of PKE and outputs the message \(m\).

Theorem 5 (Theorem 3 from [12]) The scheme PKE is IND-CCA2 secure, provided that SIG is an EUF-CMA secure one-time signature and the same requirements as in Theorem 4 are given.

V. RESEARCH PROPOSAL

The LPN problem is conjectured to be a hard problem. A major improvement in solving this problem would be a breakthrough as it would mean also a new result in the problem of decoding linear random codes. Section III presented two algorithms that solve LPN in sub-exponential time. These two, and also the other BKW variants may be structured as follows: given \(n\) queries from the LPN 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. One example of the reduction technique is the one given by Lemma 3, where we do the xor of the queries to zero a block. After the reduction techniques are applied, the algorithm runs a solving technique that recovers the secret \(s'\). Then it restarts to fully recover \(s\). Our interest is to study all the existing reduction and solving techniques used to solve the LPN problem. Comparing the existing tools may give an optimal strategy to solve LPN and can give a lower bound for solving it. Thus, for a security parameter, it can give practical values for the \(k\) and \(\tau\) parameters, in the cryptographic primitives that base their security on LPN.

Also, few research has been done in the context of LPN with a small value for the noise parameter \(\tau\). It is an open problem to say what is the boundary for \(\tau\) where the LPN problem becomes an easy problem. A small \(\tau\) gives rise to an LPN instance where the secret is sparse. We plan to study to see what are the best solving algorithms (e.g. exhaustive search, meet-in-the-middle) to recover a sparse secret vector.

LWE is a generalization of the LPN problem where we work in the field \(\mathbb{Z}_q\). That is, we are given queries of the form \((v, b)\), where \(v \leftarrow \mathbb{Z}_q^n, b = \langle v, s \rangle + \epsilon\) with \(\epsilon\) chosen from a noise distribution over \(\mathbb{Z}_q\) (usually a Gaussian noise). There exists, both quantum [26] and classical [8] reductions of LWE to hard lattice problems. This reduction result has not been shown for the LPN case. One hard task is to study this reduction by employing the right techniques for \(\mathbb{Z}_2\).

It is conjectured that LPNs is resistant to quantum computers. Given this and the fact that we can say what are the proper parameters for the LPN problem, we plan to extend existing LPN-based cryptosystems. There exists already fully-homomorphic encryption schemes based on LWE [9] and several schemes based on lattices. A fully-homomorphic scheme allows to perform any operation on ciphertexts, without requiring the knowledge of the secret key, such that the decrypted result corresponds to the output of that operation on the corresponding plaintexts. We would like to see what are the techniques that must be used in order to transform an LPN-based cryptosystem into a fully homomorphic one.

REFERENCES