# Comparing Lattice Families for Bounded Distance Decoding near Minkowski's Bound [*]

Oleksandra Lapiha

DIENS, École normale supérieure, Paris, France

August 13, 2021

**Abstract**

In this report we analyse and compare the complexity of solving the Bounded Distance Decoding problem in two families for discrete logarithm lattices. Our algorithm uses the internal structure of the lattice to decode an error close to Minkowski's bound efficiently. This procedure can be used as a decryption algorithm of an encryption scheme, where the internal structure of the lattice serves as a secret key. In addition, one of these lattices was used in [1] to construct a family of one way functions. We present cryptanalysis of the mentioned scheme and we prove that the stated size of the keys is insufficient for a required security level.

## 1 Introduction

Bounded distance decoding is one of the fundumental problems in Information theory. It arises when two parties need to communicate over a noisy channel. For example, think of a bad quality phone line. And we are concerned with the integrity of transmitted data. We are dealing with information loss

[*] These are the results of the author's internship done at CWI Amsterdam under the supervision of Léo Ducas. The internship was completed in July 2020 as a part of the Masters Program at ENS Paris.

that occurs due to the information flow being slightly disturbed by forces of nature.

To achieve integrity we encode messages as points in Euclidean space. A natural way to arrange the space of messages is to follow a Euclidean lattice (all integer linear combinations of a linearly independent set of vectors in $\mathbb{R}^n$).

The decoding procedure aims to identify the point of the lattice closest to the output point. If the disturbance in the network is small enough the closest lattice point will indeed be the input message. This will help us identify the upper bound on how much error a perfect decoding algorithm can handle. We know that the distance between lattice points is bounded from below by the length of the shortest vector(note $\lambda_1(\mathcal{L})$) of the lattice. If the error exceeds $\frac{\lambda_1(\mathcal{L})}{2}$, the closest point might not be the input message anymore so decoding becomes impossible.

We would like to have an algorithm where the decoding radius is close to $\frac{\lambda_1(\mathcal{L})}{2}$, and where $\lambda_1(\mathcal{L})$ is as large as possible. An upper bound on $\lambda_1(\mathcal{L})$ is given by Minkowski's first Theorem.

**Theorem 1** (Minkowski's First Theorem)**.** *For any full-rank lattice $\mathcal{L}$ of rank $n$ and for $l_1$ norm of the shortest vector length the following inequality holds:*

$$\lambda_1(\mathcal{L}) \leq (n!)^{1/n} \det(\mathcal{L})^{1/n} \sim \frac{n}{e} \det(\mathcal{L})^{1/n}$$

*Using the $l_1$ norm will be natural for our decoding algorithm.*

Furthermore, we know that a random lattice $\mathcal{L}$ is likely to have a $\lambda_1(\mathcal{L})$ close to this bound; yet solving the bounded distance decoding problem in random lattice is hard.

Our work focuses on two families of lattices for which the BDD problem has an efficient solution and error is close to the Minkowski's bound. The first one is a generalization of a lattice discussed in [2] and used in Chor-Rivest cryptosystem [3]. The second one was used in [1] for construction of a trapdoor function and has a similar decoding algorithm. Apart from decoding algorithms, we discuss an efficient way to compute the basis of a given lattice. It is important to be able to compute some representation of the lattice efficiently because this is a way to craft lattice points from messages we would like to transmit. We also suggest security improvements for the encryption scheme from [1] using our decoding algorithm and perform its cryptanalysis to improve their parameter selection.

**Discrete logarithm lattices.**

In this work, we will discuss many lattices of similar flavour. They are defined as a kernel of a morphism from $\mathbb{Z}^n$ to another group. Let us give you a short definition of the lattice given in [2] which serves as a basis for our first generalization.

Let us consider numbers $m$ which is a prime power, $n$ and $B$ such that we can find $n$ primes $p_1, \dots, p_n$ which do not divide $m$ and are bounded by $B$. We consider a group morphism:

$$\psi : \mathbb{Z}^n \to (\mathbb{Z}/m\mathbb{Z})^*$$

$$(x_1, \dots, x_n) \mapsto \prod_{i=1}^{n} p_i^{x_i} \pmod m$$

The kernel of $\psi$ is a subgroup of $\mathbb{Z}^n$ so it is a lattice.

$$\mathcal{L} \coloneqq \ker \psi = \{(x_1, \dots, x_n) \in \mathbb{Z}^n | \prod_{i=1}^{n} p_i^{x_i} \equiv 1 \pmod m)\}$$

**Definition 1.** *For every matrix $H \in \mathbb{Z}_q^{n \times k}$ the following defines a lattice $\mathcal{L} = \{x \in \mathbb{Z}^n : Hx \equiv 0 \pmod q\}$. This representation is called a parity check representation of the lattice $\mathcal{L}$.*

Using the properties of parity check representation it is simple to compute the basis of this lattice. It only requires the computation of $n$ discrete logarithms in $(\mathbb{Z}/m\mathbb{Z})^*$ it can be done because the multiplicative group of $\mathbb{Z}/m\mathbb{Z}$ is cyclic. Discrete logarithms can be computed in polynomial time with a combination of Pohlig-Hellman [4] and Pollard-$\rho$ [5] algorithms. It's possible as they are calculated modulo a smooth number. The generalization we give will require more work in this regard.

**Efficient decoding algorithm.**

We use a decoding algorithm discussed in [2] and [1]. We provide a few modifications for it to fit our scenario in the second part of the work, but the skeleton of the algorithm will stay the same.

We are given a lattice $\mathcal{L}$ defined above and a point $t = (t_1, \dots t_n) \in \mathbb{R}^n$ which doesn't belong to $\mathcal{L}$ and we would like to find the lattice point $x$ closest to $t$. We assume that error is a vector of real positive numbers whose norm is bounded. The algorithm has the following steps:

1. Round every coordinate to $t$ to deal only with discrete error $\lfloor t \rceil = x + \lfloor e \rceil$. Denote $e' = \lfloor e \rceil$, $t' = \lfloor t \rceil$. Lattice point $x$ is not affected by this operation since $\mathcal{L} \subset \mathbb{Z}^n$.

2. Compute $\psi(t) = \prod_{i=1}^{n} p_i^{x_i + e'_i} \pmod{m} = \prod_{i=1}^{n} p_i^{e'_i} \pmod{m}$. If the error is small enough here we recover non-reduced value $v = \prod_{i=1}^{n} p_i^{e'_i}$.

3. Recover prime number factorization of $v$ using trial division. Powers of primes will be the coordinates of our error.

4. Subtract the error from $t'$.

For negative errors we use rational number reconstruction techniques and the bound on the error will come from there. In this work by an efficient algorithm we mean an algorithm that runs in polynomial time.

**Organisation of the document.**

In the section 2 we discuss the first lattice family. We provide details on:

- how to compute its basis 2.2
- what it the complexity of this algorithm 2.3
- and how much of the error we can decode 2.4

The next part of our work 3 is dedicated to the family of polynomial lattices.

- it's basis computation is discussed in the section 3.2
- the decoding radius for this case is discussed in 3.4

Then we compare these two families with respect to basis computation and decoding algorithms in 3.5. Finally, we perform cryptanalysis of the encryption scheme presented in [1] and propose improvements to it in the section 4.

**Implementation.**

Sagemath implementation of every algorithm for basis computation and message decoding discussed in the document is available on GitHub at `https://github.com/olapiha/bounded_distance_decoding`.

# 2 Generalization of the construction for integers

In this section, we present a generalization of the decoding algorithm introduced in [2]. In their paper, Léo Ducas and Cécile Pierrot use properties of the group $(\mathbb{Z}/m\mathbb{Z})^*$ for modulus $m$ which is a prime power to decode efficiently. The decoding radius achievable that way depends on the ratio

$$\frac{\ln(m)}{\varphi(m)}$$

The higher the ratio the larger the decoding radius will be. In our generalization, we take $m'$ an arbitrary product of prime powers. If $m$ and $m'$ are of the same size, as $m'$ is smoother $\varphi(m')$ will be smaller which gives us a better decoding radius.

In the construction for $m$ prime power authors compute lattice basis directly using the fact that $(\mathbb{Z}/m\mathbb{Z})^*$ is cyclic and we can calculate discrete logarithms of its elements.

Our result is a deterministic efficient algorithm for computation of the basis for any integer $m$. We find a way to deal with the structure of the group $(\mathbb{Z}/m\mathbb{Z})^*$ in a different way and compute lattice basis through its dual.

## 2.1 Definition of discrete logarithm lattice

In this chapter we take $m = \prod_{i=1}^{k} q_j^{e_j}$ where $\{q_j\}$ are odd prime numbers and $\{e_j\}$ are positive integers. Similarly to the initial construction we take numbers $n$ and $B$ such that we can find $n$ primes $p_1, \ldots, p_n$ different from every $q_j$ and bounded by $B$. We consider a group morphism:

$$\psi : \mathbb{Z}^n \to (\mathbb{Z}/m\mathbb{Z})^*$$

$$(x_1, \ldots, x_n) \mapsto \prod_{i=1}^{n} p_i^{x_i} \pmod{m}$$

The kernel of $\psi$ is a subgroup of $\mathbb{Z}^n$ so it is a lattice. We will call it discrete logarithm lattice from now on.

$$\mathcal{L} \coloneqq \ker \psi = \{(x_1, \ldots, x_n) \in \mathbb{Z}^n | \prod_{i=1}^{n} p_i^{x_i} \equiv 1 \pmod{m}\}$$

We work with a group $(\mathbb{Z}/m\mathbb{Z})^*$ that is not necessarily cyclic anymore so we cannot exploit properties of discrete logarithm right away. Nevertheless, the Chinese Remainder Theorem (CRT) gives us the structure of the group:

$$(\mathbb{Z}/m\mathbb{Z})^* \sim \prod_{j=1}^{k} (\mathbb{Z}/q_j^{e_j}\mathbb{Z})^*$$

For every prime $q_j > 2$ and every $e_j \geq 1$ the group $(\mathbb{Z}/q_j^{e_j}\mathbb{Z})^*$ is known to be cyclic. So now, we can consider discrete logarithms in every component of the product to find a lattice basis.

Applying the CRT gives us the following equivalence

$$\mathcal{L} = \ker \psi = \{(x_1, \ldots, x_n) \in \mathbb{Z}^n | \forall 1 \leq j \leq k : \prod_{i=1}^{n} p_i^{x_i} \equiv 1 \pmod{q_j^{e_j}}\}$$

Going even further, suppose we know for every $j$ a generator $\beta_j$ of $(\mathbb{Z}/q_j^{e_j}\mathbb{Z})^*$. Using the morphism between the cyclic multiplicative group and its additive group of exponents we get representation consisting of linear relations:

$$\mathcal{L} = \{(x_1, \ldots, x_n) \in \mathbb{Z}^n | \forall 1 \leq j \leq k : \sum_{i=1}^{n} x_i \log_{\beta_j} p_i \equiv 0 \pmod{\varphi(q_j^{e_j})}\}$$

Note that discrete logarithm functions that we are using have different input and output domains

$$\forall 1 \leq j \leq k : \log_{\beta_j} : (\mathbb{Z}/q_j^{e_j}\mathbb{Z})^* \to (\mathbb{Z}/\varphi(q_j^{e_j})\mathbb{Z})$$

This is almost a parity check representation of $\mathcal{L}$ except we have many parity check type conditions with different moduli. Therefore, $\mathcal{L}$ is an intersection of lattices $\mathcal{L}_j$ where each of them is defined as a parity check lattice:

$$\mathcal{L}_j := \{(x_1, \ldots, x_n) \in \mathbb{Z}^n | \sum_{i=1}^{n} x_i \log_{\beta_j} p_i \equiv 0 \pmod{\varphi(q_j^{e_j})}\}$$

## 2.2 Computing a basis of the lattice

Our goal is to compute a basis of discrete logarithm lattice to be able to encode messages as its points afterwards. The idea of our algorithm is to compute the basis of the dual lattice first. And then obtain primal one from the dual. Let us recall a definition of a dual lattice and a dual basis.

**Definition 2.** *For a lattice $\mathcal{L} \subseteq \mathbb{R}^n$ we define $\mathcal{L}^* \subseteq (\mathbb{R}^n)^*$ as the lattice of all linear maps $f : \mathbb{R}^n \to \mathbb{R}$ such that every lattice point is mapped to an integer value.*

Linear maps can be represented as an inner product function with a fixed vector. So equivalently

$$\mathcal{L}^* = \{y \in \mathbb{R}^n | \forall x \in \mathcal{L} : \langle x, y \rangle \in \mathbb{Z}\}$$

**Definition 3.** *For a basis $B = (b_1, \dots, b_n) \in \mathbb{R}^{m \times n}$, define the dual basis $D = (d_1, \dots, d_n) \in \mathbb{R}^{m \times n}$ as the unique basis that satisfies*

- $\mathrm{Span}(D) = \mathrm{Span}(B)$
- $B^T D = I$

If B is a square matrix then $(B^T)^{-1}$ satisfies the definition. In the case of a non-square matrix, one can verify that $D = B(B^T B)^{-1}$ is the dual basis.

Our algorithm has the following steps

1. Calculate parity check representations for every $\mathcal{L}_j$
2. Get dual generating set of their intersection
3. Eliminate linear dependencies in the generating set
4. Obtain the basis of the primal lattice from the dual

We will describe each of them in more details

### 2.2.1 Parity check representations (Step 1.)

In the section 2.1 we were able to represent $\mathcal{L}$ as an intersection of $\mathcal{L}_1, \dots, \mathcal{L}_k$ for which we have parity check representations:

$$\mathcal{L}_j \coloneqq \{(x_1, \dots, x_n) \in \mathbb{Z}^n | \sum_{i=1}^{n} x_i \log_{\beta_j} p_i \equiv 0 \pmod{\varphi(q_j^{e_j})}\}$$

To compute them we calculate many discrete logarithms in finite groups. As one probably knows, it is a hard problem for a general case in the sense that we don't know a polynomial algorithm to solve it. We need to choose parameters such that we can still compute them efficiently. We discuss this choice in the section (2.3) of this document.

### 2.2.2 Dual generating set (Step 2.)

To describe this step we need two lemmas

**Definition 4.** *For two lattices $\mathcal{L}_1$, $\mathcal{L}_2$ we define their sum*

$$\mathcal{L}_1 + \mathcal{L}_2 \coloneqq \{x + y | x \in \mathcal{L}_1, y \in \mathcal{L}_2\}$$

This space $\mathcal{L}_1 + \mathcal{L}_2$ can be generated by concatenation of bases of $\mathcal{L}_1$ and $\mathcal{L}_2$. It is a sum of two additive subgroups of $\mathbb{R}^n$ so it stays an additive subgroup. But it is not always discrete so it doesn't necessarily form a lattice.

**Lemma 1.** *Suppose $\mathcal{L} = \bigcap_{j=1}^{k} \mathcal{L}_j \neq \{0\}$ and $\mathcal{L}^*$, $\mathcal{L}_j^*$ are duals of the respective lattices. Then $\mathcal{L}^* = \sum_{j=1}^{k} \mathcal{L}_j^*$ .*

*Proof.* Let us start discussing the expression on the right hand side. Take lattices $\mathcal{L}_1, \dots, \mathcal{L}_n$ and evaluate the dual of their sum

$$\begin{aligned}(\sum_{j=1}^{n} \mathcal{L}_j)^* &= \{y \in \mathbb{Z}^n | \forall x_1 \in \mathcal{L}_1, \dots, \forall x_n \in \mathcal{L}_n : \langle y, \sum_{j=1}^{n} x_j \rangle \in \mathbb{Z}\} \\ &= \{y \in \mathbb{Z}^n | \forall 1 \leq j \leq n, \forall x_j \in \mathcal{L}_j : \langle y, x_j \rangle \in \mathbb{Z}\}\end{aligned}$$

So $y$ must be an element of every $\mathcal{L}_j^*$. Therefore, $(\sum_{j=1}^{n} \mathcal{L}_j)^* = \bigcap_{j=1}^{k} \mathcal{L}_j^*$. Applying this assertion to lattices $\mathcal{L}_1^*, \dots, \mathcal{L}_n^*$ we have

$$(\sum_{j=1}^{n} \mathcal{L}_j^*)^* = \bigcap_{j=1}^{k} \mathcal{L}_j = \mathcal{L}$$

Now taking dual lattices of both sides of the equation we obtain:

$$(\sum_{j=1}^{n} \mathcal{L}_j^*)^{**} = \mathcal{L}^*$$

□

**Lemma 2.** *Let $B$ be a square matrix, $B \in \mathbb{R}^{n \times n}$. Suppose we are given parity check representation of a lattice $\mathcal{L} = \{x \in \mathbb{Z}^n | Bx \equiv 0 \pmod{p}\}$ Then **rows** of the matrix*

$$\begin{pmatrix} \frac{1}{p} \cdot B \\ I_n \end{pmatrix}$$

*form a generating set of the dual lattice.*

*Proof.* Another equivalent definition for $\mathcal{L}$ would be:

$$\mathcal{L} = \{x \in \mathbb{Z}^n | \frac{1}{p} Bx \equiv 0 \pmod 1\}$$

Therefore, can represent $\mathcal{L}$ as an intersection of the following lattices:

$$\mathcal{L}_1 = \mathbb{Z}^n$$

$$\mathcal{L}_2 = \{x \in \mathbb{R}^n | \frac{1}{p} Bx \in \mathbb{Z}^n\}$$

Then from lemma 1

$$\mathcal{L}^* = (\mathcal{L}_1 \cap \mathcal{L}_2)^* = \mathcal{L}_1^* + \mathcal{L}_2^*$$

It is obvious that $(\mathbb{Z}^n)^* = \mathbb{Z}^n$. To prove that $(\frac{1}{p}B)^T$(here basis vectors are columns) is a basis of the dual it is enough to show $(\frac{1}{p}B)^{-1}$ is basis of the primal lattice. This is quite simple:

$$\forall x \in \mathbb{Z}^n : \frac{1}{p} B \cdot (\frac{1}{p} B)^{-1} \cdot x \in \mathbb{Z},$$

which proves the inclusion in one direction. For the inclusion in the opposite direction we have:

$$\forall x \in \mathcal{L} : \frac{1}{p} B \cdot x = y \in \mathbb{Z}^n \implies x = (\frac{1}{p} B)^{-1} \cdot y, y \in \mathbb{Z}^n$$

A generating set of the sum of lattices can be obtained by concatenation of their bases, so we obtain our desired result. □

The basis computation algorithm takes parity check representations of every lattice $\mathcal{L}_j$ scales them and adds an identity matrix. The output is the concatenation of calculated generating sets.

### 2.2.3 Eliminating linear dependencies using elementary matrix transformations (Step 3.)

Now we have obtained a generating set of the dual lattice but we would like to have its basis. The idea is to transform the matrix to its row echelon form so the resulting set has some zero vectors which we will discard, and obtain the basis of the lattice by keeping the remaining non-zero generators.

Out of all elementary matrix transformations, we are only allowed adding to a row an integer multiple of another row, interchanging two rows, multiplying a row by -1. These three possibilities are called unimodular transformations. As one may have noticed the only restriction is that we can't multiply a row by an integer different from ±1. This would result in a sublattice of $\mathcal{L}$ with a higher determinant.

We use an algorithm for reducing the matrix to its Hermite normal form. It is an equivalent of row echelon form for matrices over $\mathbb{Z}$ with a restriction to unimodular transformations. Our input matrix can have rational coefficients so we first transform them into integers multiplying by the least common multiple of all denominators. We divide by it when the matrix is in the Hermite form.

Resulting vectors might be quite long. If we want to control the size of the basis we can use the LLL algorithm instead. This change will, for example, make our encoded messages shorter and easier to decode.

### 2.2.4 Primal basis from dual basis (Step 4.)

A way to calculate a basis of primal lattice having the basis of the dual is straightforward having its definition that we stated here 3. If $D$ denotes a dual basis, then $B = D(D^T D)^{-1}$ is a primal basis.

## 2.3 Complexity analysis

The first step (1) of the basis computation algorithm (2.2) boils down to many computations of discrete logarithms in a finite group. For every distinct prime factor $q_j^{e_j}$ of the modulus we compute $n$ discrete logarithms(one for every prime $p_i$). It is $n \cdot k$ iterations in total. Let's refer to $(\mathbb{Z}/q_j^{e_j}\mathbb{Z})^*$ as group $G$. Group order is equal to $|G| = \varphi(q_j^{e_j}) = q_j^{e_j-1}(q_j - 1) = \prod_{i=1}^{k} t_i^{a_i}$. It is a $q$-smooth integer, so to efficiently calculate discrete logarithms in this group we can use a combination of Pohlig-Hellman [4] and Pollard-$\rho$ [5] algorithms. Overall complexity in group operations is

$$O(\sum_{i=1}^{k} a_i(\ln(|G|) + \sqrt{t_i}))$$

To be polynomial in lattice dimension, $a_i \leq e_j - 1$ and $t_i \leq q_j$ need to be polynomial in $n$. As we have $n \cdot k$ such operations, $k$ should also be polynomial in $n$.

It is easy to see that step 2 takes linear time in $n + k$. Step 3 is Hermite normal form reduction which has the same complexity as Gaussian elimination so runs in time polynomial in $n + k$.

Finally, step 4 includes matrix multiplication and inversion for matrices whose dimension is bounded by $n + k$. They take polynomial time.

## 2.4 Decoding radius

Decoding algorithm extends naturally to the generalized setting. Let us remind that normalized decoding $l_1$-radius was equal to

$$\bar{r}_1 = \frac{\ln(m/2)}{4\varphi(m)^{1/n} \cdot \ln(B)}$$

where $B$ is a constant bounding all $p_i$

We still haven't configured parameters $k, q_j, p_i$. The primes $q_j, p_i$ must be pairwise different and as small as possible, so we need $n + k$ distinct prime numbers. We choose $p_i$ to be equal to first $n$ prime numbers and $q_j$ equal to the next $k$ smallest primes that we haven't used yet.

The more factors $m$ the smoother $\varphi(m)$ will be, but also the ratio $\frac{ln(m/2)}{\varphi(m)}$ decreases as $m$ tends to infinity. For every dimention $n$ we take $m = \prod_{j=1}^{l} q_j$ where $l$ we be determined by a script as the first local maximum of the radius.

## 2.5 Was it all worth it?

As it turns out factorization of $m$ does not affect the decoding radius substantially. It can only introduce a minor improvement. We prove that with the following lemma:

**Lemma 3.** *Assume the primes $p_1, \ldots, p_n$ are the same for both lattices. If $a \cdot m \sim b \cdot m' \sim e^n$, when $n \to \infty$ Then*

$$\frac{\bar{r}_m^{(1)}}{\bar{r}_{m'}^{(1)}} \to 1, n \to \infty$$

*Proof.* The proof is in the appendix A □

Refer to the figure 1 to compare the initial construction and its generalization in practice.

From a practical perspective, such complications may indeed not be Worthy. But we've definitely learned a thing or two on our way.

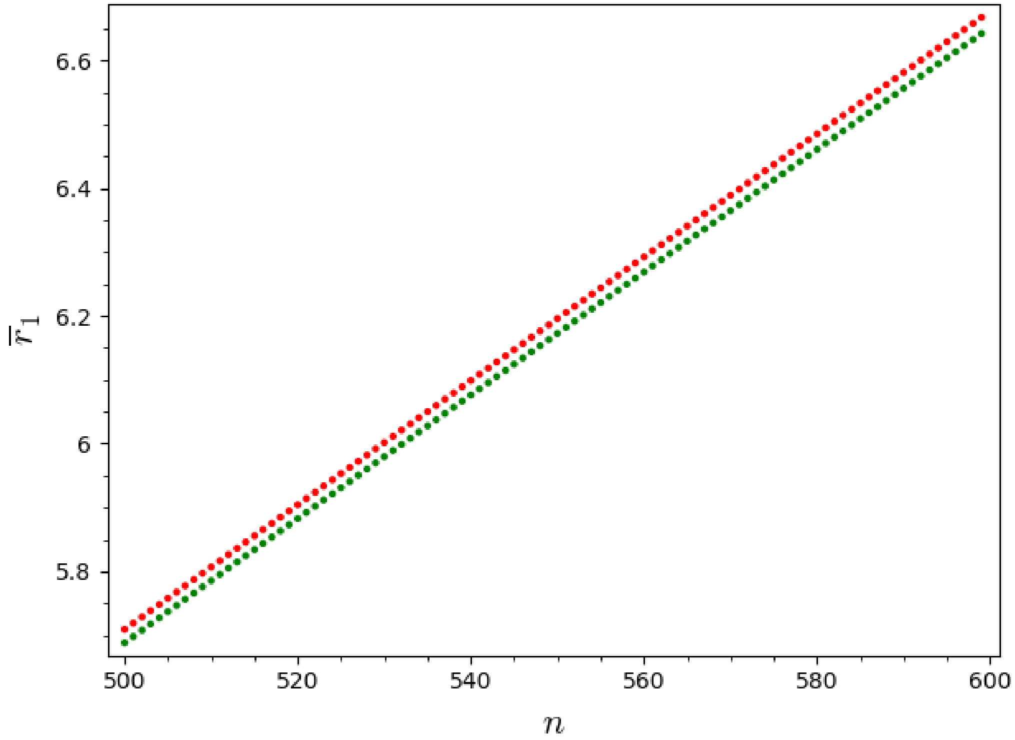

Figure 1: Comparison of the normalized decoding radius for initial construction(in green) and the generalization(in red).

# 3 New Construction for Polynomial Lattices

## 3.1 Definition of Polynomial Lattices

Let us set parameters a prime power $q$ and integers $k$, $d$ and $n$. Let $\mathbb{F}_q[x]$ be polynomial ring over a field $\mathbb{F}_q$. We take a set of $k$ irreducible polynomials $c_j(x) \in \mathbb{F}_q[x]$, $j = 1, ..., k$ of degree $d$. According to the analogue of the prime number theorem for polynomials [6] $k$ cannot be greater than $\frac{q^d}{d}$.

Define $c(x) \coloneqq \prod_{j=1}^{k} c_j(x)$. We are going to work in the multiplicative group of the quotient ring of $\mathbb{F}_q[x]$ with respect to $c(x)$. Chinese Remainder Theorem helps to determine the structure of $\left(\mathbb{F}_q[x]/c(x)\right)^*$:

$$\left(\mathbb{F}_q[x]/c(x)\right)^* \sim \prod_{i=1}^{k} \left(\mathbb{F}_q[x]/c_i(x)\right)^* \sim \prod_{i=1}^{k} \mathbb{F}_{q^d}^*$$

Every component is a quotient by an irreducible polynomial, therefore it's a field. The multiplicative group of a field is cyclic, so we can consider discrete logarithms in every component of the product to find a lattice basis.

Consider a vector $\alpha = (\alpha_1, ..., \alpha_n) \in \mathbb{F}_q^n$ where the $\alpha_i$'s are pairwise different and not roots of $c(\cdot)$.

Now consider a group morphism:

$$\psi : \mathbb{Z}^n \to \left(\mathbb{F}_q[x]/c(x)\right)^*$$

$$(u_1, ..., u_n) \mapsto \prod_{i=1}^{n} (x - \alpha_i)^{u_i} \pmod{c(x)}$$

Similarly to previous constructions, the lattice is defined as the kernel of the morphism $\psi$:

$$\mathcal{L} = \ker\psi = \{(u_1, ..., u_n) \in \mathbb{Z}^n | \prod_{i=1}^{n} (x - \alpha_i)^{u_i} \equiv 1 \pmod{c(x)}\}$$

We will be calling $\mathcal{L}$ the polynomial lattice from now on. Applying CRT gives us an equivalent definition for $\mathcal{L}$:

$$\mathcal{L} = \ker\psi = \{(u_1, ..., u_n) \in \mathbb{Z}^n | \forall 1 \le j \le k : \prod_{i=1}^{n} (x - \alpha_i)^{u_i} \equiv 1 \pmod{c_j(x)}\}$$

Supposing we know $\beta_j$ a generator of $\left(\mathbb{F}_q[x]/c_j(x)\right)^*$ for every $j$ we get another representation by computing discrete logarithms in the multiplicative group of every component:

$$\mathcal{L} = \{(u_1, ..., u_n) \in \mathbb{Z}^n | \forall 1 \leq j \leq k : \sum_{i=1}^{n} u_i \log_{\beta_j}(x - \alpha_i) \equiv 0 \pmod{q^d - 1}\}$$

What might cuase a confusion is that each $\log_{\beta_j}$ has a different input domain. For every $j$: $\log_{\beta_j}$ maps $\left(\mathbb{F}_q[x]/c_j(x)\right)^*$ to $(\mathbb{Z}/(q^d - 1)\mathbb{Z})$.

## 3.2 Lattice computation for polynomials

We obtained a parity check representation of $\mathcal{L}$. To calculate a basis of $\mathcal{L}$ we can follow a simplified version of the algorithm for discrete logarithm lattices. We obtain the dual basis by scaling the parity check matrix and concatenating it with $I_n$. Then we remove linear dependencies and finally obtain primal basis from the dual. So basis computation algorithm has the following steps:

1. Obtain the parity check representation of $\mathcal{L}$.
2. Transform it into the basis of $\mathcal{L}^*$.
3. Recover the primal basis from the dual.

### 3.2.1 Parity check representation 1.

To compute the parity check representation of the polynomial lattice we need to compute $\forall 1 \leq i, j \leq n : log_{\beta_j}(x - \alpha_i) \pmod{q^d - 1}$ and form them into a matrix. The order of multiplicative group is $q^d - 1$ which is not necessarily a smooth integer so we cannot use Pohlig-Hellman [4] and Pollard-$\rho$ [5] approach to compute them efficiently. We are going to choose $q^d = n^{O(1)}$ so group order is overall polynomial in the lattice dimension and use Pollard-$\rho$ [5] algorithm to compute discrete logarithms.

### 3.2.2 Dual basis 2.

Lemma 2 gives us a straightforward way to compute the generating set of the dual from a parity check matrix. To obtain its basis we eliminate linear dependencies by reducing the generating matrix to its Hermite normal form as in the section 2.2.3.

### 3.2.3 Recovering Primal basis from Dual 3 (Theory).

In this section, $Q$ is an arbitrary integer, not related to the lattices constructed before.

Lattices that admit a parity check representation are called $q$-ary lattices. They lie between $\mathbb{Z}_q^n$ and $\mathbb{Z}^n$: $\mathbb{Z}_q^n \subset \mathcal{L} \subset \mathbb{Z}^n$. For this kind of lattices, it's possible to transform the parity check lattice into its basis efficiently. In the algorithm, we will rely on the parity check matrix to have a special shape.

**Definition 5.** *A matrix $A \in \mathbb{R}^{n \times m}$ is in the systematic form if it has the following form:*

$$A = \left[ I_n | A' \right]$$

*Where $I_n$ stands for an identity matrix of dimensions $n \times n$ and $A'$ is an arbitrary matrix of dimensions $n \times (m-n)$*

**Lemma 4.** *Suppose $\mathcal{L}$ is defined by a parity check matrix. $\mathcal{L} = \{x \in \mathbb{Z}^n : Hx \equiv 0 \pmod q\}$ and $H \in \mathbb{Z}_q^{k \times n}$ is in the systematic form $H = [I_k|D]$. Then rows of $G = [-D^T|I_{n-k}]$ are generating vectors of the lattice $\mathcal{L}$ reduced modulo $q$.*

*Proof.* Let us first prove that every vector generated with the matrix $G$ belongs to the lattice.

$$\forall x : H \cdot G^T x = [I_k|D] \cdot [-D^T|I_{n-k}]^T x = (D - D)x = 0 \pmod q$$

Indeed we see that $G$ generates a sublattice of $\mathcal{L}$, let us call it $\mathcal{L}'$. If $G$ generates $\mathcal{L}'$ modulo $q$ then $G' = \left[ \begin{array}{c|c} -D^T & I_{n-k} \\ \hline qI_k & 0 \end{array} \right]$ generates all of the lattice. $det(G') = det(\mathcal{L}') = q^k$.

It is left to prove that $\mathcal{L}$ has the same determinant, then $\mathcal{L} = \mathcal{L}'$ .

Parity check matrix defines the lattice as a kernel of a linear transformation defined by $H$.

$$\phi : \mathbb{Z}^n \to \mathbb{Z}_q^n$$
$$x \mapsto Hx \pmod q$$

Since $H$ is in the systematic form, the image contains a space equivalent to $\mathbb{Z}_q^k$ so it contains at least $q^k$ points. $\mathcal{L}$ is a sublattice of $\mathbb{Z}^n$ so its determinant can be computed as

$$det(\mathcal{L}) = |\mathbb{Z}^n/\mathcal{L}| \cdot det(\mathbb{Z}^n) = |\mathbb{Z}^n/ker(\phi)|$$

Using the first isomorphism theorem we have $|\mathbb{Z}^n/ker(\phi)| = |Im(\phi)| \geq q^k$. So

$$q^k \leq Im(\phi) = det(\mathcal{L}) \leq det(L') = q^k$$

Therefore $\mathcal{L}' = \mathcal{L}$. QED. □

**Corrolary 1.** *Suppose the matrix $H$ is in permuted systematic form e.g.$\exists P-$ column permutation matrix $P \cdot H = [I|D]$. Then $P \cdot G$ are generating vectors of the lattice $\mathcal{L}$ reduced modulo $q$.*

#### 3.2.4 Recovering Primal basis from Dual 3 (Practice).

Let us build an algorithm specific to the polynomial lattices. First, in our case the lattice is $Q$-ary for $Q = q^d - 1$. Here $q$ is a prime power - the modulus of the base field $\mathbb{F}_q$.

At the start of this step of lattice computation, we are given the parity check matrix if the polynomial lattice. The lattice is the set of solutions to a homogeneous equation modulo $q^d - 1$ defined by the parity check matrix, therefore, none of the elementary transformations modulo $q^d - 1$ can change it. So we can apply Gaussian elimination to reduce it to the systematic form.

Here $q^d - 1$ is not a prime number, so some of the elements of $H$ might not be invertible modulo $q^d - 1$, and we cannot render them equal to one. That is why it is useful to have permuted systematic form. We can then try to invert elements of a particular row until we find an invertible one. In practice the algorithm is able to reduce every matrix to its systematic form, so we make an assumption that there are not so many elements we cannot invert.

Using Lemma 4 we can obtain the set of generating vectors $G$ of the polynomial lattice $\mathcal{L}$ modulo $q$. Since we are dealing with the q-ary lattice, to obtain the basis of the $\mathcal{L}$ itself we need to add the basis of $\mathbb{Z}_q^n$ and remove resulting linear dependencies. It is not strictly necessary to perform these steps as having $G$ we can craft points of the lattice $\mathcal{L}$.

Let us put all the steps together in an algorithm:

1. Reduce the parity check matrix to its permuted systematic form.
2. Transform it into the generating matrix modulo $q$.
3. Concatenate with $q \cdot I_n$.
4. Reduce obtained matrix to its Hermite normal form.

## 3.3 Decoding algorithm modification

We could check now if the framework of the decoding algorithm developed before fits this new setting. In this section, we discuss the modification we need to make and the decoding radius we can achieve with this approach. Modifying the algorithm to treat polynomials instead of integers leads to the following steps.

1. Round every coordinate to $t$ to deal only with discrete error $\lfloor t \rceil = v + \lfloor e \rceil$. Note $e' = \lfloor e \rceil$, $t' = \lfloor t \rceil$.
2. Compute $\psi(t) = \prod_{i=1}^{n}(x - \alpha_i)^{v_i + e'_i} \pmod{c(x)} = \prod_{i=1}^{n}(x - \alpha_i)^{e'_i} \pmod{c(x)}$.
3. Reconstruct the numerator $n$ and the denominator $d$ of $\prod_{i=1}^{n}(x - \alpha_i)^{e'_i}$ that correspond to positive and negative parts of the error.
4. Recover prime number factorization of $n$ and $d$ using trial division. Powers of primes will be the coordinates of our error.
5. Subtract just recovered error from $t'$.

Except for the step 3, all other steps translate directly to the case of polynomials. Step 3 needs more work, let us discuss rational function reconstruction in more detail.

### 3.3.1 Rational function reconstruction 3

In the work of [7] authors study rational function reconstruction in depth. Given $g, f$ the goal is to find $n, d \in \mathbb{F}[x]$ that $\deg(n) + \deg(d) < \frac{\deg(f)}{2}$ and $\frac{n}{d} = g \pmod{f}$ The main algorithm the discuss is based in Extended Euclidean Algorithm(EEA). We perform extended euclidean division and the select a step where the degree of the current quotient is greater or equal to $\frac{\deg f}{2}$. The authors refer to it as Wang's algorithm. Here is its formal description.

Here function $lc()$ outputs the leading coefficient of the input polynomial.

The correctness of this algorithm follows from the following two lemmas. The first one proves that under certain conditions one of the rows of EEA contains the correct $n$ and $d$. The second lemma tells us how to select the correct row.

**Algorithm 1** Rational Function Reconstruction.

```
procedure Reconstruct($f, g$)
    $r_0 = f$ $r_1 = g$
    $t_0 = 0$ $t_1 = 1$
    $q = 1$
    while $\deg(q) \leq \frac{\deg(f)}{2}$ do
        $q = r_0 // r_1$
        $(r_0, r_1) = (r_1, r_0 - q r_1)$
        $(t_0, t_1) = (t_1, t_0 - q t_1)$
    end while
    if $\gcd(r_0, t_0) \neq 1$ or $\deg(r_0) + \deg(t_0) \geq \frac{\deg(f)}{2}$ then
        return FAIL
    else:
        return $(\frac{r_0}{lc(t_0)}, \frac{t_0}{lc(t_0)})$
    end if
end procedure
```

**Lemma 5.** *Let $\mathbb{F}$ be a field, $f, g, r, s, t \in \mathbb{F}[x]$ with $r = sf + tg$, $t \neq 0$, $\deg(f) > 0$, and $\deg(r) + \deg(t) < \deg(f)$. Suppose $r_i, s_i, t_i$ for $0 \leq i \leq l+1$ be the elements of the ith iteration in the Extended Euclidean Algorithm(EEA) for $f$ and $g$ (e.i. $r_i = s_i f + t_i g$).*
*Then there exists a nonzero element $\alpha \in \mathbb{F}[x]$ such that $r = \alpha r_j$, $s = \alpha s_j$, $t = \alpha t_j$, where $\deg(r_j) \leq \deg(r) < \deg(r_{j-1})$*

*Proof.* Find the proof in [8] (Lemma 5.15 page 116) □

So if the solution exists it must be one of the pairs $(r_i, t_i)$ of the EEA.

**Lemma 6.** *Let $\mathbb{F}$ be a field $f, g, n, d \in \mathbb{F}[x]$ are polynomials such that $lc(d) = 1$, $gcd(n, d) = gcd(f, d) = 1$ and $g = fracnd \pmod f$. Let $j$ be the index of a quotient with maximal degree in the EEA for $f$ and $g$. If $\deg(f) > 2(\deg(n) + \deg(d))$ the $j$ is unique and $n = r_j$, $d = t_j$*

*Proof.* Find the proof in [7] (Lemma 2.3 page 186) □

### 3.3.2 Factorization by trial division

Input: A polynomial $g$ such that $\deg(g) \leq m$ whose roots are among $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$

Output: $e_1, \dots, e_n$ s.t. $g = \prod_{i=1}^{n}(x-\alpha_i)^{e_i}$
There's only $n$ possible roots, one trial division takes $O(m)$ time and the number of factors is bounded by $m$. So overall complexity is $O(m^2 n)$.

## 3.4 Decoding radius

Rational function reconstruction puts an upper bound on how much error this algorithm can decode. Let us first consider a simpler case where every coordinate of the error is positive, so the step 3 becomes trivial. The goal of this section is to find the parameters which provide maximal normalized decoding radius for arbitrary discrete error and calculate this upper bound.

### 3.4.1 Only positive discrete error

Suppose we receive $t = u + e$ where $u \in \mathcal{L}$, $\|e\|_1 \le r_1$ and $\forall i : e_i \in \mathbb{N}$. Then we can compute

$$\prod_{i=1}^{n}(x-\alpha_i)^{t_i} = \prod_{i=1}^{n}(x-\alpha_i)^{u_i}\prod_{i=1}^{n}(x-\alpha_i)^{e_i} \pmod{c(x)}$$

If $\|e\|_1 = \sum_{i=1}^{n} e_i \le \deg(c) = d \cdot k$ the operation above will give us exactly the polynomial $\prod_{i=1}^{n}(x-\alpha_i)^{e_i}$. Then we can recover $e_i$, $1 \le i \le n$ from the factorization.
So $l_1(r_1) = d \cdot k$.

Due to the nature of the bound above it is natural to talk about the length $r_1$ in $l_1$ norm.

### 3.4.2 Arbitrary discrete error

Now we have $\forall i : e_i \in \mathbb{Z}$. Then

$$\prod_{i=1}^{n}(x-\alpha_i)^{t_i} \pmod{c(x)} = \prod_{i=1}^{n}(x-\alpha_i)^{e_i} = \frac{\prod_{i\in I}(x-\alpha_i)^{e_i}}{\prod_{j\in J}(x-\alpha_j)^{-e_j}}$$

**Lemma 7.** *Given $g, c$ where $\deg(c) = d \cdot k$ we can recover $f_1, f_2 \in \mathbb{F}[x]$ that $\forall i = 1; 2$: $\deg(f_i) \le \lfloor \frac{dk}{2} \rfloor$ and $\frac{f_1}{f_2} = g \pmod{c}$ in polynomial time.*

So we can decode every message for which $\|e\|_1 = \sum_{i=1}^{n} |e_i| \le \lfloor \frac{dk}{2} \rfloor$

If we know more about the shape of the error the upper bound becomes equal to the case with only positive error. We can reconstruct $g = \frac{n}{d} \pmod f$ if it is known that $\deg(n) < N$, $\deg(d) < D$ where $N + D < \deg(f)$ refer to [7] for more information.

### 3.4.3 Normalized radius

From the bound on the error we already have we obtain the mormalized raduis $\bar{r}_1 = \frac{dk}{2 \cdot det(\mathcal{L})^{1/n}}$ where $det(\mathcal{L}) = \Phi(c(x)) = (q^d - 1)^k$.

$$\bar{r}_1 = \frac{dk}{2 \cdot (q^d - 1)^{k/n}}$$

Let us optimize the values of $d$ and $k$.

We have the following constraints:

1. $q^d = n^{O(1)}$,
   so that computing discrete logarithm in polynomial time (Section 3.2.1).
2. $dk < q^d$,
   so we can find enough irreducible polynomias of degree $d$ (Section 3.1).
3. $n \leq q$,
   so we can take $n$ polynomials $x - \alpha_i$ from $\mathbb{F}_q[x]$ (Section 3.1).

From these constraints we can immediately conclude that $d$ must be constant.

To obtain not more than a logarithmic gap from Minkowski's bound we need the following asymptotics: $q = a \cdot n$, $d = b$, $k = c \cdot \frac{n}{\log(n)}$. What we are going to do next is finding optimal values for parameters $a, b$ and $c$. We want to maximize the following function:

$$\bar{r}_1 = \frac{bc \cdot n}{2\log(n)((an)^b - 1)^{c/\log(n)}} \sim \frac{bc \cdot n}{2e^{bc}\log(n)}$$

Parameter $a$ by constraint 3 in 3.4.3 should be greater or equal to 1.

$$\frac{bc \cdot n}{2\log(n)((an)^b - 1)^{c/\log(n)}} \leq \frac{bc \cdot n}{2\log(n)(n^b - 1)^{c/\log(n)}}$$

The choice of $a$ is independent from choices of $b$ and $c$ so we can configure it to the smallest possible value $a = 1$.

The function we are considering with fixed parameters $c, b$ tend to the same value as $\frac{bc \cdot n}{2\log(n) n^{cb/\log(n)}}$. To simplify the analysis we will find the best parameters for the latter.

Note $e := bc$, $f_n(e) = \frac{e \cdot n^{1-\frac{e}{\log(n)}}}{2\log(n)}$. We would like to prove that for any value of $n$, $argmax(f_n(e)) = 1$

$$\begin{aligned} f_n'(e) &= \frac{n^{1-\frac{e}{\log(n)}}}{2\log(n)} + \frac{e \cdot n^{1-\frac{e}{\log(n)}} \cdot \log(n)\big(-\frac{1}{\log(n)}\big)}{2\log(n)} \\ &= \frac{n^{1-\frac{e}{\log(n)}}}{2\log(n)}(1-e) \end{aligned}$$

The value $e = 1$ is the only solution to the equation $f_n'(e) = 0$. It is easy to verify that it corresponds to the maximum of $f_n(e)$ and it doesn't depend on the value of $n$.

Therefore, $e = b \cdot c = 1$. In practice we fix $b = 2, c = \frac{1}{2}$. In the end, the decoding radius our algorithm achieves is

$$\bar{r}_1 = \frac{n}{2 \cdot \log(n)(n^2-1)^{1/2\log(n)}}$$

### 3.5 Comparing the decoding radius of the algorithms.

For the first family final normalized error radius that the algorithm can handle is:

$$\bar{r}_1 = \frac{\ln(m/2)}{4 \cdot \varphi(m)^{1/n} \cdot \ln((n+1)\ln(n+1))}$$

As we can see on the plot 2 for the second construction the radius is much better:

$$\bar{r}_1 = \frac{n}{2 \cdot \log(n)(n^2-1)^{1/2\log(n)}}$$

But both of them are still logarithmically far from Minkowski's bound.

## 4 Cryptanalysis of the LLXY17 cryptosystem.

Let us give a high-level description of the encryption scheme presented in [1]. It is based on the family of polynomial lattices that we described above. The public key of Alice is a matrix $G$ that generates a sublattice of a polynomial

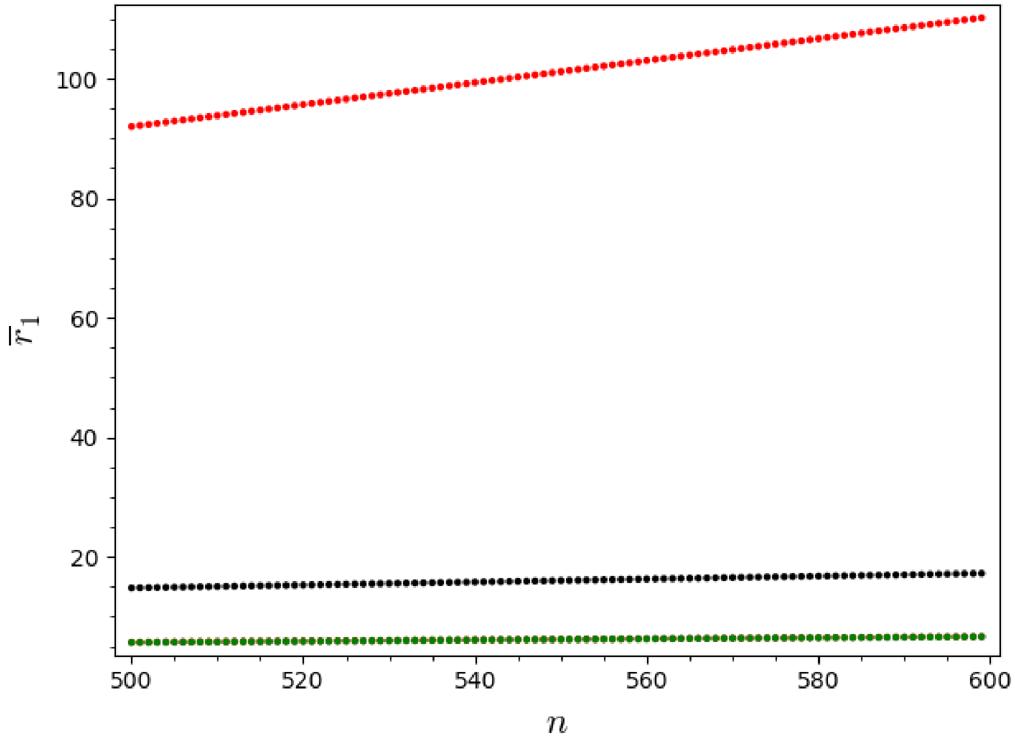

Figure 2: Comparison of all algorithms and theoretical upperbound (in red): basic algorithm for integer lattice (in green), algorithm for polynomial lattices (in blue).

lattice $\mathcal{L}$. When Bob wants to send a message $m$ to Alice he generates a random noise $e \in \{0,1\}^n$ of hamming weight $w$ and sends the ciphertext $c = Gm + e$.

Alice's private key contains parameters $c(x)$ and $(\alpha_1, \dots, \alpha_n)$ used to construct the lattice $\mathcal{L}$. To decode the message Alice performs error decoding algorithm discussed in section 3.3. As we saw, for the decoding procedure to work $w$(the $l_1$ norm of $e$) should be less than the upperbound on the error.

We are given the ciphertext $c$ we would like to obtain the message without the key. Having the generating matrix $G$ we can obtain the parity check representation $H$ of $\mathcal{L}$. Multiplying the ciphertext by $H$ we obtain:

$$H \cdot c = H \cdot Gm + H \cdot e = H \cdot e$$

We would like to try every possible value of the error until $H \cdot c = H \cdot e$. If we can learn the value of the error, we can subtract it from the ciphertext and obtain the message multiplying by $G^{-1}$.

Let us consider three different approaches to make the bruteforce attack more efficient.

We compute the cost of every attack in the appendix B

## 4.1 Information Set Decoding Attack

We take a vector $y \in \mathbb{Z}_q^k$, a matrix $A \in \mathbb{Z}_q^{k \times n}$. It can be reduced to its permuted systematic form $H = U \cdot A = [I_k|D]$.

The goal is to find a vector $x \in \{0,1\}^n$ with small Hamming weight $|x| = w$ that

$$Ax = y$$

Or equivalently $Hx = U \cdot Ax = Uy =: t$, here we can use our knowledge about the shape of $H$.

In this attack we partition $x$ on two vectors $x_1 \in \{0,1\}^k$ and $x_2 \in \{0,1\}^{n-k}$ so we have

$$t = x_1 + D \cdot x_2$$

We make a bet of the weight partition between $|x_1| = w_1$ and $|x_2| = w_2$, where $w_1 + w_2 = w$. Now we enumerate only the possible values of $x_2$, compute $x_1 = t - D \cdot x_2$ and check if it satisfies $|x_1| = w_1$. If we don't find a correct pair with this weight distribution, we rerandomize H and t and start over. The pseudo-code is given as Algorithm 2.

## 4.2 Meet in the Middle

In this attack, we have the same goal but no information about the form of the matrix A. We partition $x$ and $A$ on two equal parts: $A = [A_1|A_2]$, $x = (x_1|x_2)$. Then $Ax = y$ is equivalent to

$$A_1x_1 + A_2x_2 = y$$

If we can find vectors $x_1$ and $x_2$ for which values $A_1x_1$ and $y - A_2x_2$ collide their concatenation $x = (x_1|x_2)$ will satisfy $Ax = y$. Here we make a bet that weight is distributed equally between the two parts. For all values of $x_1$ with correct weight we compute the value $A_1x_1$ and put $x_1$ into a hash-table with the index $h(A_1x_1)$, where $h(\cdot)$ is an arbitrary suitable hash function. Then for every $x_2$ we look up the value with index $h(y - A_2x_2)$ in the table. If the cell is not empty we found the correct pair. Similarly, the pseudo-code of the attack is given below as Algorithm 3.

## 4.3 Combining the two approaches.

Now we would like to combine Information Set Decoding attack with Meet in the Middle approach to make it even more efficient.

We return to the case when $A$ is reduced to the systematic form $H = U \cdot A = [I_k|D_1|D_2]$ we partition $x = (x_0|x_1|x_2)$ on three vectors $x_0 =\in \{0,1\}^k$, $x_1, x_2 \in \{0,1\}^{\frac{n-k}{2}}$. Then

$$Ax = x_0 + D_1x_1 + D_2x_2$$

We make bet that $|x_0| = w_1$, $|x_1| = |x_2| = \frac{w_2}{2}$ and perform Meet-in-the-Middle attack. Now we would like to find **approximate** collisions between $D_1x_1$ and all possible $t - D_2x_2$

For that we desing a compression function $f$ that will often map close vectors to the same value. It operates as follows:

$$\forall v = (v_1, \dots, v_k) \in \mathbb{Z}_q^k : f_p(v) = (\lfloor v1/p \rceil, \lfloor v2/p \rceil, \dots, \lfloor vk/p \rceil)$$

Here $p$ is a parameter of the function.

We store a table of $f(D_1x_1)$ in memory and look-up there for $t - D_2x_2$. We will need to deal with false positives and false negatives in this collision search. Too many false positives can increase the cost of the search for every randomization of the public key matrix and too many false negatives can increase the number of randomizations themselves. To ease the analysis the pseudo-code is given as Algorithm 4.

## 4.4 Adding negative errors

Our algorithm allows to decode errors of both signs efficiently, so adding it will improve security of [LLXY17] encyption scheme. Let us estimate how resistant is the scheme agains our most efficient attack when it uses $e \in \{0,1\}^n$ and $e \in \{0,1,-1\}^n$ of the same $l_1$ norm. In the ternary case the error contains the same number of positive and negative coordinates so that we can provide the same decoding radius as for the positive error. The costs and improvement are presented in the following table.

| Cost of attacks for different parameters | | | | | |
|---|---|---|---|---|---|
| Parameters | Bruteforce | ISD | MitM | IDS+MitM | imrovement rate |
| n = 100.0<br>k = 10.0<br>w = 9.0<br>q = 100.0 | $2^{48}$ | $2^{39}$ | $2^{27}$ | $2^{25}$ | |
| Ternary | $2^{57}$ | $2^{41}$ | $2^{31}$ | $2^{28}$ | $2^{3}$ |
| n = 500.0<br>k = 40.0<br>w = 30.0<br>q = 500.0 | $2^{171}$ | $2^{132}$ | $2^{92}$ | $2^{81}$ | |
| Ternary | $2^{201}$ | $2^{137}$ | $2^{107}$ | $2^{92}$ | $2^{11}$ |
| n = 1000.0<br>k = 72.0<br>w = 50.0<br>q = 1000.0 | $2^{295}$ | $2^{224}$ | $2^{155}$ | $2^{137}$ | |
| Ternary | $2^{345}$ | $2^{230}$ | $2^{180}$ | $2^{155}$ | $2^{18}$ |

## 4.5 LLXY17 parameter selection

We would also like to comment on parameter selection made for the scheme. In the following table, we present costs of the attack for the smallest and largest parameters suggested by [1] for their encryption scheme. Their estimate security level for the first one is $2^{106}$ and $2^{119}$. With our best attack, we are able to break it with an average cost of $2^{25}$ and $2^{26}$ which is feasible in practice.

| Cost of attacks for LLXY17 parameters | | |
|---|---|---|
| Parameters | Claimed security | IDS+MitM cost |
| n = 230.0<br>k = 201.0<br>w = 28.0<br>q = 263.0 | $2^{106}$ | $2^{25}$ |
| n = 260.0<br>k = 228.0<br>w = 31.0<br>q = 293.0 | $2^{119}$ | $2^{26}$ |

In conclusion, the parameters taken for this scheme need to be seriously reconsidered.

# 5 Future work

The main question left to answer is what are the correct parameters to use in the scheme of [1]. For that, we need to take into account the cost of our most efficient attack, the attacks discussed in the original paper and the constraints of our decoding algorithm. After the parameters are computed we need to see if the scheme is still practical.

In the attack developed for the ternary case we do not take into account that the error has to contain the same number of positive and negative coordinates, so our security analysis can be improved for the ternary errors.

# 6 Acknowledgements

We would like to thank Léo Ducas for suggesting the topic of this project and for supervising it. We also thank Koen de Boer for helpful discussions. This research was supported in part by the Cryptology Group of the Centrum Wiskunde & Informatica and in part by École normale supérieure de Paris.

# A Proof of Lemma 3

*Proof.* Since the primes are the same, they are also bounded by the same constant $B$ so the term $\frac{1}{4\ln(B)}$ cancels out in the numerator and denominator.

$$\begin{aligned}\frac{\bar{r}_m^{(1)}}{\bar{r}_{m'}^{(1)}} &= \frac{\ln(m/2)\cdot\varphi(m')^{1/n}}{\ln(m'/2)\cdot\varphi(m)^{1/n}} \\ &= \frac{\ln(e^n/2a)\cdot\varphi(m')^{1/n}}{\ln(e^n/2b)\cdot\varphi(m)^{1/n}} \\ &= \frac{\varphi(m')^{1/n}}{\varphi(m)^{1/n}}\end{aligned}$$

We apply a few classic results to estimate the growth of $\varphi(n)$ [9]:

$$\limsup_{n\to\infty}\frac{\varphi(n)}{n} = 1$$

$$\liminf_{n\to\infty}\frac{\varphi(n)\cdot\ln\ln(n)}{n} = e^{-\gamma}$$

where $\gamma$ is the Euler constant, $e^{-\gamma} = 0.56145948\ldots$

$$\begin{aligned}\lim_{n\to\infty}\frac{\varphi(m')^{1/n}}{\varphi(m)^{1/n}} &\leq \frac{\limsup\limits_{n\to\infty}\varphi(m')^{1/n}}{\liminf\limits_{n\to\infty}\varphi(m)^{1/n}} \\ &= \lim_{n\to\infty}\left(\frac{m\cdot\ln\ln(m')}{m'}\right)^{1/n} \\ &= \lim_{n\to\infty}(\ln\ln(e^n/b))^{1/n} = 1\end{aligned}$$

□

# B Calculating the cost of attacks

## B.1 ISD: Binary case

The average cost of such an algorithm can be calculated as

$$T_{w_2} = \frac{x_2 \text{ bruteforce cost}}{Pr(|x_2| = w_2||x| = w)}$$

To compute the numerator will go through the steps of the algorithm 2:

**Algorithm 2** ISD attack

```
 1: procedure DECODING($A, y$)
 2:     Reduce A it's randomized systematic form: $(I_k|D)$
 3:     for $x_2$ of weight $w_2$ do
 4:         compute $x_1 = t - Dx_2$
 5:         if $|x_1| = w_1$ and contains only $0, 1$ then
 6:             return Derandomize $x = (x_1, x_2)$
 7:         end if
 8:     end for
 9:     goto line 2.
10: end procedure
```

- Step 3-4: $O(k \cdot n^2)$
- Step 5: $O(k \cdot (n-1) \cdot (k-1))$
- Step 6-11: $O(k \cdot w_2 \cdot \binom{n-k}{w_2})$

In total we have: $k \cdot (n^2 + (n-1)(k-1) + w_2 \binom{n-k}{w_2})$ Computing the denominator is a simple task as well:

$$Pr(|x_2| = w_2||x| = w) = \frac{\binom{n-k}{w_2} \cdot \binom{k}{w_1}}{\binom{n}{w}}$$

Therefore

$$T_{w_2} = \frac{k \cdot \binom{n}{w} \cdot (n^2 + (n-1)(k-1) + w_2 \binom{n}{w_2})}{\binom{k}{w_1}\binom{n-k}{w_2}}$$

## B.2 ISD: Ternary case

Now if we select the errors from $\{0, 1, -1\}^n$(we will be calling them *ternary errors* from now on), the only thing that changes is the number of choices for $x_2$. The numerator:

$$k \cdot (n^2 + (n-1)(k-1) + w_2 \binom{n-k}{w_2} \cdot 2^{w_2})$$

The denominator:

$$Pr(|x_2| = w_2||x| = w) = \frac{\binom{n-k}{w_2} \cdot \binom{k}{w_1}}{\binom{n}{w}}$$

So,

$$T_{w_2} = \frac{k \cdot \binom{n}{w} \cdot (n^2 + (n-1)(k-1) + w_2\binom{n}{w_2} \cdot 2^{w_2})}{\binom{k}{w_1}\binom{n-k}{w_2}}$$

To minimize the average cost we take an optimal value of $w_2$ calculated with a script.

## B.3 MitM: Binary case

**Algorithm 3** MitM attack

```
 1: procedure DECODING($A, y$)
 2:     Generate random matrix $U$
 3:     $(A_1|A_2) \leftarrow A \cdot U$
 4:     $t = y \cdot U$
 5:     for $x_1$ of weight $\frac{w}{2}$ do
 6:         compute $A_1 x_1$
 7:         store in the hash table
 8:     end for
 9:     for $x_2$ of weight $\frac{w}{2}$ do
10:         compute $t - A_2 x_2$
11:         look up in the hash table for a collision
12:         if collision found then
13:             return $x = (x_1, x_2)$
14:         end if
15:     end for
16:     goto line 2.
17: end procedure
```

Here we bet that the weight is distributed equally on both sides. So the average cost can be calculated as follows:

$$T = \frac{\text{cost of finding a collision}}{Pr(|x_1| = |x_2| = w/2 \,||x| = w)}$$

Similarly to the previous attack we compute the cost of running one iteration of the algorithm 3:

- Step 2: $n * k * \log(q)$

- Step 3-4: $O(k \cdot n^2)$
- Step 5-8: $O(k \cdot \frac{w}{2} \cdot \binom{n/2}{w/2})$
- Step 9-15: $O(k \cdot \frac{w}{2} \cdot \binom{n/2}{w/2})$

The time cost we have in total is equal to

$$T = O(k \cdot (n^2 + w\binom{n/2}{w/2}))$$

Also, we need to store every $x_1$. So, the cost in terms f memory is equal to

$$M = \binom{n/2}{w/2} \cdot \log(n) \cdot \frac{w}{2}$$

Computing the value of the denominator we obtain:

$$Pr(|x_1| = |x_2| = w/2) = \frac{\binom{n/2}{w/2}^2}{\binom{n}{w}}$$

The total average time cost:

$$T = \frac{k \cdot \binom{n}{w} \cdot (n^2 + w\binom{n/2}{w/2})}{\binom{n/2}{w/2}^2}$$

## B.4 MitM: Ternary case

Numerator(time):

$$T = k \cdot (n^2 + w \cdot \binom{n/2}{w/2} \cdot 2^{w/2})$$

Memory:

$$M = \binom{n/2}{w/2} \cdot 2^{w/2} \cdot \log(n) \cdot \frac{w}{2}$$

Denominator:

$$Pr(|x_1| = |x_2| = w/2) = \frac{\binom{n/2}{w/2}^2}{\binom{n}{w}}$$

Total average time cost:

$$T = \frac{k \cdot \binom{n}{w} \cdot 2^{w/2} \cdot (n^2 + w\binom{n/2}{w/2})}{\binom{n/2}{w/2}^2}$$

## B.5 ISD+MitM: Binary case

**Algorithm 4** ISD+MitM attack

```
 1: procedure DECODING($(A, y)$)
 2:     Reduce A it's randomized systematic form: $(I_k|D_1|D_2)$
 3:     for $x_1$ of weight $\frac{w_2}{2}$ do
 4:         compute $f_p(D_1 x_1)$
 5:         store in the hash table
 6:     end for
 7:     for $x_2$ of weight $\frac{w_2}{2}$ do
 8:         compute $f_p(t - D_2 x_2)$
 9:         look up in the hash table for a collision
10:         if collision found then
11:             compute $D_1 x_1$
12:             compute $x_0 = t - D_2 x_2 - D_1 x_1$
13:             if $|x_0| = w_1$ and contains only $0, 1$ then
14:                 return Derandomize $x = (x_0, x_1, x_2)$
15:             end if
16:         end if
17:     end for
18:     goto *line 2.*
19: end procedure
```

Let us first compute the cost of running the algorithm when we were lucky to generate such matrix $U$ that we end up finding a vector $x$ that satisfies all the coditions. The steps refer to the algorithm 4.

- Step 3-4: $O(k \cdot n^2)$
- Step 5: $O(k \cdot (n-1) \cdot (k-1))$
- Step 6-9: $O(k \cdot \frac{w_2}{2} \cdot \binom{(n-k)/2}{w_2/2})$
- Step 10-19: $O(k \cdot \frac{w_2}{2} \cdot \binom{(n-k)/2}{w_2/2} + k \cdot \{\#\text{ collisions}\})$

In total we have: $O(k\cdot(n^2+(n-1)(k-1)+w_2\binom{(n-k)/2}{w_2/2}++E(\{\#\text{ positives}\})))$

Concerning memory cost, we only store the hash table of $x_1$ in a cell with index $f_p(D_1 x_1)$ for every $x_1$. The weight of $x_1$ is small so we only remember the set of nonzero coordinates $\log(n) \cdot \frac{w_2}{2}$. Number of such $x_1$ is equal to $\binom{(n-k)/2}{w_2/2}$

So the overall complexity is $\binom{(n-k)/2}{w_2/2} \cdot \log(n) \cdot \frac{w_2}{2}$

The total average cost can be computed with a formula:

$$T = \frac{k \cdot (n^2 + (n-1)(k-1) + w_2\binom{(n-k)/2}{w_2/2} + E(\{\# \text{ collisions}\}))}{Pr(|x_0| = w_1, |x_1| = |x_2| = \frac{w_2}{2} ||x| = w_1 + w_2) Pr(\text{there was no false negative collision})}$$

Let us calculate missing parts of the formula:

$$\begin{aligned} E(\{\# \text{ collisions}\}) &= \binom{n}{w_2} \cdot Pr(\text{collision}) \\ &= \binom{n}{w_2} \cdot \frac{1}{\lfloor q/p \rceil}^{n} \end{aligned}$$

To simplify the notations for the computation of false negatives the approximate collision search between $y_1 \coloneqq D_1 x_1$ and $y_2 \coloneqq t - D_2 x_2$ let us call $T$ the hash table of $y_1$'s. The number of boxes to which we map $y_1$ and $y_2$ with our compression function we denote with $b = \lfloor q/p \rceil$. We assume that $y_1, y_2$ are distributed uniformly over $\mathbb{Z}_q^k$ because otherwise we would be able to obtain some information on $x_1$ and $x_2$ and speed up the bruteforce which contradicts our security assumption. $t_i$ signify the non-zero coordinates of $x_0$

$$\begin{aligned} Pr_{y_2}(\text{false negative}) &= Pr_{y_2}(\exists y_1 \in T : f_p(y_1) \neq f_p(y_2), |x_0| \coloneqq |y_2 - y_1| = w_1, x_0 \in \{0,1\}^k) \\ &= Pr_{y_2}(\exists x_0 s.t. |x_0| = w_1, x_0 \in \{0,1\}^k : f_p(y_2 - x_0) \neq f_p(y_2), y_2 - x_0 \in T) \\ &\leq (\text{union bound}) \leq \sum_{x_0} Pr_{y_2}(f_p(y_2 - x_0) \neq f_p(y_2), y_2 - x_0 \in T) \\ &\leq (\text{union bound}) \leq \sum_{x_0} \sum_{i=1}^{w_1} Pr_{y_2}((y_2)_{t_i} \in \{kp + 1p = 0, \dots b\}, y_2 - x_0 \in T) \\ &= \sum_{x_0} \sum_{i=1}^{w_1} \frac{\binom{n-k/2}{w_2/2} \cdot \frac{b}{q}}{q^k} \\ &= \sum_{x_0} w_1 \cdot \frac{\binom{n-k/2}{w_2/2} \cdot \frac{b}{q}}{q^k} \\ &= \binom{k}{w_1} \cdot w_1 \cdot \frac{\binom{n-k/2}{w_2/2} \cdot \frac{b}{q}}{q^k} \end{aligned}$$

The probability of getting a correct weight distribution is:

$$Pr(|x_0| = w_1, |x_1| = |x_2| = \frac{w_2}{2}) = \frac{\binom{n-k/2}{w_2/2}^2 \binom{k}{w_1}}{\binom{n}{w}}$$

Putting everything together we obtain the total average cost.

## B.6 ISD+MitM: Ternary case

Note that we do **not** take into account that the error has to contain the same number of positive and negative coordinates, so the attack we present can be optimized even further.

Computational cost:

- Step 3-4: $O(k \cdot n^2)$
- Step 5: $O(k \cdot (n-1) \cdot (k-1))$
- Step 6-9: $O(k \cdot \frac{w_2}{2} \cdot \binom{(n-k)/2}{w_2/2} \cdot 2^{w_2/2})$
- Step 10-19: $O(k \cdot \frac{w_2}{2} \cdot \binom{(n-k)/2}{w_2/2} \cdot 2^{w_2/2} + k \cdot \{\# \text{ positives}\})$

In total we have: $O(k \cdot (n^2 + (n-1)(k-1) + w_2 \binom{(n-k)/2}{w_2/2} \cdot 2^{w_2/2} + E(\{\# \text{ collisions}\})))$

Memory cost:

$$M = \binom{(n-k)/2}{w_2/2} \cdot 2^{w_2/2} \cdot \log(n) \cdot \frac{w_2}{2}$$

Total average cost:

$$T = \frac{k \cdot (n^2 + (n-1)(k-1) + w_2 \binom{(n-k)/2}{w_2/2} \cdot 2^{w_2/2} + E(\{\# \text{ collisions}\}))}{Pr(|x_0| = w_1, |x_1| = |x_2| = \frac{w_2}{2} \,|\, |x| = w_1 + w_2) Pr(\text{there was no false negative collision})}$$

Missing parts of the formula :

$$E(\{\# \text{ collisions}\}) = \binom{n}{w_2} \cdot 2^{w_2/2} \cdot Pr(\text{collision}) = \binom{n}{w_2} \cdot 2^{w_2/2} \cdot \frac{1}{\lfloor q/p \rceil}^{n}$$

$$\begin{aligned}
Pr_{y_2}(\text{false negative}) &= Pr_{y_2}(\exists y_1 \in t : f_p(y_1) \neq f_p(y_2), |x_0| \coloneqq |y_2 - y_1| = w_1, x_0 \in \{0,1\}^k) \\
&= Pr_{y_2}(\exists x_0 s.t. |x_0| = w_1, x_0 \in \{0,1\}^k : f_p(y_2 - x_0) \neq f_p(y_2), y_2 - x_0 \in T) \\
&\leq \text{(union bound)} \leq \sum_{x_0} Pr_{y_2}(f_p(y_2 - x_0) \neq f_p(y_2), y_2 - x_0 \in T) \\
&\leq \text{(union bound)} \leq \sum_{x_0} \sum_{i=1}^{w_1} \mathbb{1}_{(x_0=1)} Pr_{y_2}((y_2)_{t_i} \in \{kp + 1p = 0, \dots b\}, \\
&y_2 - x_0 \in T) + \mathbb{1}_{(x_0=-1)} Pr_{y_2}((y_2)_{t_i} \in \{kp - 1p = 0, \dots b\}, y_2 - x_0 \in T) \\
&= \sum_{x_0} \sum_{i=1}^{w_1} \frac{\binom{n-k/2}{w_2/2} \cdot \frac{b}{q}}{q^k} \\
&= \sum_{x_0} w_1 \cdot \frac{\binom{n-k/2}{w_2/2} \cdot \frac{b}{q}}{q^k} \\
&= \binom{k}{w_1} \cdot 2^{w_1} \cdot w_1 \cdot \frac{\binom{n-k/2}{w_2/2} \cdot \frac{b}{q}}{q^k}
\end{aligned}$$

$$Pr(|x_0| = w_1, |x_1| = |x_2| = \frac{w_2}{2}) = \frac{\binom{n-k/2}{w_2/2}^2 \binom{k}{w_1}}{\binom{n}{w}}$$

Putting everything together we obtain the total average cost.

# References

[1] Z. Li, S. Ling, C. Xing, and S. L. Yeo., "On the closest vec-tor problem for lattices constructed from polynomials and their cryptographic applications.," *Cryptology ePrint Archive*, 2017.

[2] L. Ducas and C. Pierrot, "Polynomial time bounded distance decodingnear minkowski's boundin discrete logarithm lattices," *Des. Codes Cryptogr.*, pp. 87(8): 1737–1748, 2019.

[3] B. Chor and R. R. Rivest, "A knapsack-type public key cryptosystem based on arithmetic in finite fields.," *IEEE Trans. Information Theory*, pp. 34(5):901–909, 1988.

[4] S. C. Pohlig and M. E. Hellman, “An improved algorithm for computing logarithms over $gf(p)$ and its cryptographic significance,” *IEEE Transactions on Information Theory*, p. 24(1):106–110, 1978.

[5] J. Pollard, “Monte carlo methods for index computations mod p,” *Mathematics of Computation*, p. 918–924, 1978.

[6] C. F. Gauss and A. A. Clarke, *Disquisitiones Arithmeticae.* 1965.

[7] S. Khodadad and M. Monagan, “Fast rational function reconstruction,” *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC*, pp. 184–190, 2006.

[8] J. G. Joachim von zur Gathen, *Modern Computer Algebra.* 3 ed., 2013.

[9] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers.* 6 ed., 2009.