Comparing Lattice Families for Bounded Distance Decoding near Minkowski's Bound

Oleksandra Lapiha

August 7, 2020

1 Introduction

Bounded Distance Decoding problem.

Message decoding is a problem that arises when two parties need to communicate over a noisy channel. Here we are in the setting where we are only concerned with the integrity of transmitted data and we assume there's no adversary listening and modifying messages in the network. We are dealing with information loss that occurs due to our information flow being *slightly disturbed* by forces of nature.

To achieve that we encode messages as points in Euclidean space. Now we can model the rate of "disturbance" as distance between the input and output points. And call it "slight" if it can be bounded by a constant. A natural way to arrange the space of messages is 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($\lambda_1(\mathcal{L})$) 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 $\frac{\lambda_1(\mathcal{L})}{2}$. Unfortunately, the Lattice Shortest Vector Problem(SVP) is believed to be hard in the general case, so we cannot evaluate exactly how good our decoding radius is. Nevertheless, we have an upper-bound on the length of the shortest vector. So we aim to get a decoding radius close to it instead.

Theorem 1 (Minkowski's First Theorem)**.** *For any full-rank lattice $\mathcal{L}$ of rank n,*

$$\lambda_1(\mathcal{L}) \leq \sqrt{n}\det(\mathcal{L})^{1/n}$$

An instance of such problem is called a Bounded Distance Decoding problem and it is considered a hard problem for a random lattice and decoding radius of $\frac{\lambda_1(\mathcal{L})}{2}$ in the norm l_∞ [1].

Our work focuses on two families of lattices for which 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 [4] 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.

In this work we generalize the lattice family from [2] and suggest an efficient way to compute it. The decoding algorithm of the same article extends to our case directly. The second lattice family was discussed in [4] we extend the decoding algorithm devised for them to handle more types of error. We also suggest security improvements for the encryption scheme from [4] 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 $p_1, \ldots, p_n$ which do not divide m and are bounded by B. We consider a group morphism:

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

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

It is easy to see that the image will indeed be an element of $(\mathbb{Z}/m\mathbb{Z})^*$ due to our parameter selection. The kernel of ψ 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 $H \in \mathbb{Z}_q^{n\times m}$ 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.*

Using the properties of parity check repr it is simple to compute the basis of this lattice it only requires computation of n discrete logarithms in $(\mathbb{Z}/m\mathbb{Z})^*$ it can be done because the multiplicative group of a field $\mathbb{Z}/m\mathbb{Z}$ is cyclic. Discrete logarithms can be computed in polynomial time as they are calculated modulo a smooth number. More on that you can find in [2]. The generalization we give will require more work in this regard.

Efficient decoding algorithm.

We use a decoding algorithm discussed in [2] and [4]. 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 numbers whose norm is bounded. The algorithm has the following building blocks:

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

- 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 $v = \prod_{i=1}^{n} p_i^{e'_i}$.

- Reconstruct the numerator n and the denominator d of v that correspond to positive and negative parts of the error. Recover prime number factorization of n and d using trial division. Powers of primes will be the coordinates of our error.

- Subtract just recovered error from t'.

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 detail 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
- ToDo: maybe rename sections so they follow the same pattern in the second part.

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 [4] 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 via link: TBA.

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. The decoding radius achievable in polynomial-time depends on the ratio

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

The higher the ratio the larger the achievable 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:

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

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

The kernel of ψ 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 $(\mathbb{Z}/m\mathbb{Z})^*$ which is not 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.

[1]Since $p_1, \ldots, p_n$ are relatively prime with m for every element of $\mathbb{Z}^n$ its image is indeed a part of $(\mathbb{Z}/m\mathbb{Z})^*$.

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 β_j of $(\mathbb{Z}/q_j^{e_j}\mathbb{Z})^*$. Using the morphism between the cyclic group and group of exponents we get a representation in terms of conditions on the exponents:

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

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 at the same time. Therefore, $\mathcal{L}$ is an intersection of lattices $\mathcal{L}_j$ where each of them is defined as

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

2.2 Computing lattice basis

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 a 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 a 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. [2]

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 (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 you probably know it is a hard problem for a general case of which no polynomial algorithm is known. We need to choose parameters such that we can compute them efficiently. We discuss it in the section (2.3) of this document.

[2]Since columns of B are basis vectors, B is a full column rank matrix. Therefore, $B^T B$ is always invertible.

2.2.2 Dual generating set (2)

To justify 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 discreet so it doesn't necessarily form a lattice.

Lemma 1. [3] *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 from the end. Take lattices $\mathcal{L}_1, \dots, \mathcal{L}_n$ which is the dual of their sum. [4]

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

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.

[3] Check it again with fresh head

[4] We can take all x_i equal to 0 but 1 of them

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:

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

And the other way:

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

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. Output is the concatenation of calculated generating sets.

2.2.3 Eliminating linear dependencies using elementary matrix transformations (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.

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 you 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 (4)

A way to calculate a basis of primal lattice having the basis of the dual is straightforward having its definition.

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 [5] and Pollard-ρ [6] 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 be polynomial in n.

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

Finally, step four (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. 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 ratio $\frac{ln(m/2)}{\varphi(m)}$ decreases as m tends to infinity, so to make use of the difference in the ratio in the generalization, we would like m to have the same size as in the original construction. We would like to have $e_i \sim \frac{\ln(3)\cdot n}{\ln(q)\cdot k}$, but it has to be an integer, so we round this value. Then we have:

$$m = \prod_{i=1}^{k} q_i^{round(\frac{\ln(3)\cdot n}{\ln(q)\cdot k})} \sim 3^n$$

The only parameter left to configure is k. The way we set up e_is number of factors will only increase the ratio $\frac{ln(m/2)}{\varphi(m)}$. So the more factors we have the better.

Let us compare initial construction, its generalization and theoretical upper bound with respect to the decoding 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*

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

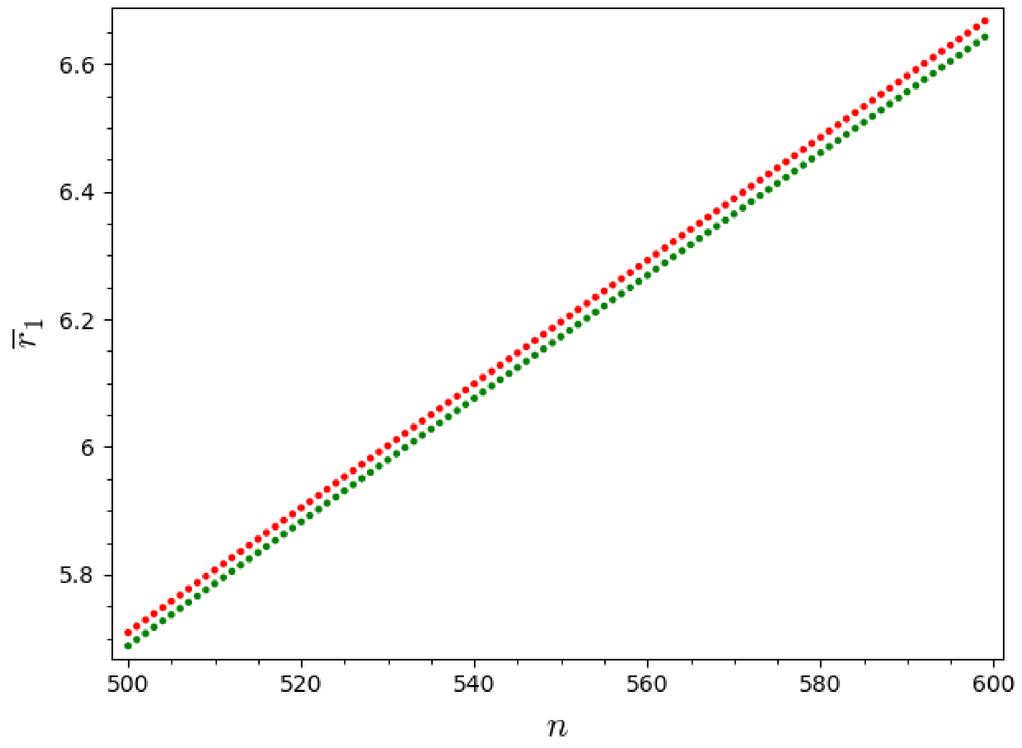

Figure 1: Comparison of the normalized decoding radius for initial construction(in green) and the generalization(in yellow). Here we took k = 20.

Proof. The proof is in the appendix A □

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 k must not 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 quotient by an irreducible polynomial, therefore it's a field. 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 α_is are pairwise different. Since polynomials $c_j(\cdot)$ are irreducible over $\mathbb{F}_q$ neither of α_i can be their root. So for all α_i we have: $c(\alpha_i) \neq 0$.

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 ψ:

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

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

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

Supposing we know β_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: $\log_{\beta_j}$ acts from $\left(\mathbb{F}_q[x]/c_j(x)\right)^*$ into $(\mathbb{Z}/(q^d - 1)\mathbb{Z})$.

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 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 [5] and Pollard-ρ [6] 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-ρ [6] 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 noraml form as in the section 2.2.3.

3.2.3 Recovering Primal basis from Dual 3 (Theory).

In this section q is and arbitrary integer, not related to the lattices cnstructed 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. Let us take an arbitrary vector x and prove that $G^T \cdot x$ belongs to $\mathcal{L}$.

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

If G generates the lattice modulo q then $G' = \left[\begin{array}{c|c} -D^T & I_{n-k} \\ \hline qI_k & 0 \end{array}\right]$ □

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 $q \to q^d - 1$. Now q means 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 trasformations 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's not so many element 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 nessesary to perfom these steps as having G we can craft points of the lattice $\mathcal{L}$.

Let us put all the steps together in an algorthm:

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 developped before wil fit 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 polinomials 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 translates 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 xcurrent quotient is greater or equal to $\frac{deg f}{2}$. The authors refer to it as Wang's algorithm. Here is its formal description.

Algorithm 1 RFR

```
1: procedure RECONSTRUCT($f, g$)
2:     $r_0 = f$ $r_1 = g$
3:     $t_0 = 0$ $t_1 = 1$
4:     $q = 1$
5:     while $deg(q) \leq \frac{deg(f)}{2}$ do
6:         $q = r_0 // r_1$
7:         $(r_0, r_1) = (r_1, r_0 - qr_1)$
8:         $(t_0, t_1) = (t_1, t_0 - qt_1)$
9:     end while
10:    if $GCD(r_0, t_0) \neq 1$ or $deg(r_0) + deg(t_0) \geq \frac{deg(f)}{2}$ then
11:        return FAIL
12:    else:
13:        return $(\frac{r_0}{lc(t_0)}, \frac{t_0}{lc(t_0)})$
14:    end if
15: end procedure
```

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

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.

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: $e_1, \ldots, e_n$ 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 upperbound on how much error this algorithm can decode. Let us first consider a simpler case where every coordinate of the error in 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 the this upperbound.

3.4.1 Only positive discrete error

Suppose we receive $t = u + e$ where $u \in \mathcal{L}$, $\|e\|_1 \leq r_1$ 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 $\|e\|_1 = \sum_{i=1}^{n} e_i \leq deg(c) = d \cdot k$ the operation above will give us exactly the polynomial $\prod_{i=1}^{n}(x-\alpha_i)^{e_i}$. Then we can recover e_i, $1 \leq i \leq n$ 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 $\forall i = 1; 2$: $deg(f_i) \leq \lfloor \frac{dk}{2} \rfloor$ and $\frac{f_1}{f_2} = g \pmod{c}$ in polynomial time.*

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

If we know more about the shape of the error the upper bound becomes even better. We can reconstruct $g = \frac{n}{d} \pmod{f}$ if it is known that $degn < N$, $degd < D$ where $N + D < degf$ 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. $q^d = n^{O(1)}$
2. $dk < q^d$
3. $n \leq q$

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

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

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

Therefore, $b \cdot c = 1$. 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:

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

And for the second construction it 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 logatihmically far from Minkowski's bound. The figure 2 will help you to see the difference in practice.

4 LLXY17 Cryptanalysis

Let us give a high level description of the encryption scheme presented in [4]. It is based on the family of polynomial lattices that we described above. The public key of Alice is a matrix G that enerates a sublattice of a polynomial 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 containes parameters $c(x)$ and $(\alpha_1, \ldots, \alpha_n)$ used to construct the lattice $\mathcal{L}$. To decode the message Alice performs error decoding algorithm discussed in the 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 given the ciphertext c we would like to obtain the message without the key. Having the generating matrx 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 evey 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 cipher text 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

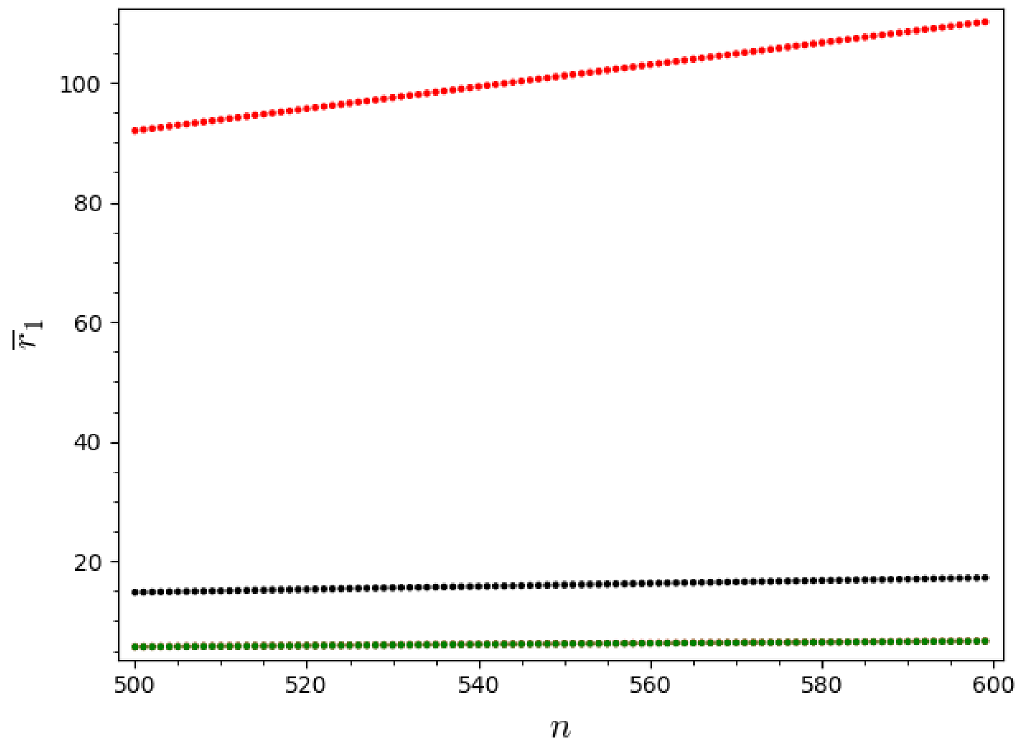

Figure 2: Comparison of all algorithms and theoretical upperbound(in red): basic algorithm for integer lattice(in green), algorithm for polynomial lattices(in blue)

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. Here we present the pseudoce of the attack. 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
```

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 $y - A_2x_2$ coincide. Here we make a bet that weight is distributed equall between the two parts.Similarly we present the pseudocode of the attack below 3.

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 $A_1x_1$
 7:         store in the hash table
 8:     end for
 9:     for $x_2$ of weight $\frac{w}{2}$ do
10:         compute $t - A_2x_2$
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
```

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 we present pseudocode of the algorithm. 4

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 $f_p(D_1x_1)$
 5:         store in the hash table
 6:     end for
 7:     for $x_2$ of weight $\frac{w_2}{2}$ do
 8:         compute $f_p(t - D_2x_2)$
 9:         look up in the hash table for a collision
10:         if collision found then
11:             compute $D_1x_1$
12:             compute $x_0 = t - D_2x_2 - D_1x_1$
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
```

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. The costs and imprevement are presented in the following table.

Cost of attacks for different parameters					
Parameters	Bruteforce	ISD	MitM	IDS+MitM	imrovement rate
n = 100.0 k = 10.0 w = 5.0 q = 100.0	4.517251e+09	2.960902e+07	2.116580e+05	2.366108e+05	
Ternary	1.445520e+11	6.725278e+07	1.197318e+06	3.293097e+05	2
n = 100.0 k = 10.0 w = 9.0 q = 100.0	1.902232e+14	2.801449e+11	8.084476e+07	2.006729e+07	
Ternary	9.739427e+16	1.699224e+12	1.829308e+09	1.854533e+08	2^3
n = 500.0 k = 40.0 w = 30.0 q = 500.0	1.792122e+51	3.963162e+39	3.737340e+27	2.087190e+24	
Ternary	1.924276e+60	1.091922e+41	1.224651e+32	3.433996e+27	2^{11}
n = 1000.0 k = 72.0 w = 50.0 q = 1000.0	3.473881e+88	2.056657e+67	3.262500e+46	1.362168e+41	
Ternary	3.911243e+103	8.987320e+68	1.094713e+54	3.432411e+46	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 [4] for their encryption scheme. Their estimate seccurity level for the first one is 2^{106} and 2^{119}. With our best attack we are able to break it with 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	in binary

n = 230.0
k = 201.0
w = 28.0
q = 263.0
	2^{106}	2.778036e+07	2^{25}

n = 260.0
k = 228.0
w = 31.0
q = 293.0
	2^{119}	4.523610e+07	2^{26}

In concusion 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 [4]. 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 parameter are computed we need to see if the scheme is still practical.

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

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

□

B Calculating the cost of attacks

B.1 ISD: Binary case

The average cost of such algorithm can be calculated as

$$T = \frac{x_2 \text{ bruteforce cost}}{Pr(w_2 \text{ is a correct bet on the weight of } x_2)}$$

Let us compute the values above. The numerator:

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

$$\begin{aligned}Pr(w_2 \text{ is a correct bet on the weight of } x_2) &= Pr(|x_2| = w_2 \,|\, |x| = w) \\ &= \frac{\binom{n-k}{w_2}\cdot \binom{k}{w_1}}{\binom{n}{w}}\end{aligned}$$

Therefore

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

B.2 ISD: Ternary case

The numerator: Only the number of choices for x_2 change

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

The denominator:

$$\begin{aligned} Pr(w_2 \text{ is a correct bet on the weight of } x_2) &= Pr(|x_2| = w_2 || x| = w) \\ &= \frac{\binom{n-k}{w_2} \cdot \binom{k}{w_1}}{\binom{n}{w}} \end{aligned}$$

Therefore

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

To minimize the average cost we take an optimal value of w_1 calculated with a script.

B.3 MitM: Binary case

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

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

Numerator:

- 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: $O(k \cdot \frac{w}{2} \cdot \binom{n/2}{w/2})$

In total:
TIME COST $= O(k \cdot (n^2 + w\binom{n/2}{w/2}))$
We need to store every x_1
MEMORY COST $= \binom{n/2}{w/2} \cdot log(n) \cdot \frac{w}{2}$
Denominator: $Pr(|x_1| = |x_2| = w/2) = \frac{\binom{n/2}{w/2}^2}{\binom{n}{w}}$
Total cost(only time):

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

B.4 MitM: Ternary case

Numerator:
TIME COST $= k \cdot (n^2 + w \cdot \binom{n/2}{w/2} \cdot 2^{w/2})$
We need to store every x_1
MEMORY COST $= \binom{n/2}{w/2} \cdot 2^{w/2} \cdot log(n) \cdot \frac{w}{2}$
Denominator: $Pr(|x_1| = |x_2| = w/2) = \frac{\binom{n/2}{w/2}^2}{\binom{n}{w}}$
Total cost(only time):

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

B.5 ISD+MitM: Binary case

Computational cost: Let us first compute the cost of running the algorithm when we were lucky to generate such matrix U that we end up finding vector x that satisfies all the coditions.

- 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: $O(k \cdot \frac{w_2}{2} \cdot \binom{(n-k)/2}{w_2/2} + k \cdot \{\#\text{ positives}\})$

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

Memory cost: We only store in memory the hash table of x_1 in a cell with index $f_p(D_1x_1)$ 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: $\binom{(n-k)/2}{w_2/2}$

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

Total average cost:

$$T = \frac{k \cdot (n^2 + (n-1)(k-1) + w_2\binom{(n-k)/2}{w_2/2} + E(\{\# \text{ positives}\}))}{Pr(\text{weight distribution bet was successful, there was no false negative in the collision search})}$$

Let us calculate missing parts of the formula:

$$E(\{\#\text{positives}\}) = \binom{n}{w_2} \cdot Pr(\text{positive})$$

$$\begin{aligned} Pr(\text{positive}) &= Pr(\text{collision happens}) \\ &= \frac{1}{\lfloor q/p \rceil}^n \end{aligned}$$

To simplify the notations for the computation of false negatives $y_1 := D_1x_1$, $y_2 := t - D_2x_2$ let us call T the hash table of y_1s. The number of boxes($\lfloor q/p \rceil$) we denote with b. 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 would break our security assumption(THINK AGAIN ABOUT THIS). t_i signify non-zero coordinates ofx_0

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

$$\begin{aligned}
Pr(\text{weight distribution is correct}) &= Pr(|x_0| = w_1, |x_1| = |x_2| = \frac{w_2}{2}) \\
&= \frac{\binom{n-k/2}{w_2/2}^2 \binom{k}{w_1}}{\binom{n}{w}}
\end{aligned}$$

Putting everything together we obtain the total average cost.

B.6 ISD+MitM: Ternary case

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

Memory cost: We only store in memory 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: $\binom{(n-k)/2}{w_2/2} \cdot 2^{w_2/2}$

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

Total average cost:

$$T = \frac{k \cdot (n^2 + (n-1)(k-1) + w_2 \binom{(n-k)/2}{w_2/2} \cdot 2^{w_2/2} + E(\{\# \text{ positives}\}))}{Pr(\text{weight distribution bet was successful, there was no false negative})}$$

Let us calculate missing parts of the formula:

$$E(\{\# \text{ positives}\}) = \binom{n}{w_2} \cdot 2^{w_2/2} \cdot Pr(\text{positive})$$

$$\begin{aligned} Pr(\text{positive}) &= Pr(\text{collision happens}) \\ &= \frac{1}{\lfloor q/p \rceil}^{n} \end{aligned}$$

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

$$\begin{aligned} Pr(\text{weight distribution is correct}) &= Pr(|x_0| = w_1, |x_1| = |x_2| = \frac{w_2}{2}) \\ &= \frac{\binom{n-k/2}{w_2/2}^2 \binom{k}{w_1}}{\binom{n}{w}} \end{aligned}$$

Putting everything together we obtain the total average cost.

References

[1] V. Lyubashevsky and D. Micciancio, "On bounded distance decoding,unique shortest vectors,and the minimum distance problem," *Advances in Cryptology - CRYPTO*, pp. 577–594, 2009.

[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] 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.

[5] 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.

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

[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.