Analysis of a Lattice-Code-based Cryptosystem & a variant of the Generalised Birthday Problem

Pabasara Athukorala

Supervisors:
Prof Steven Galbraith
Dr Jurij Volcic

Department of Mathematics
The University of Auckland

A THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN MATHEMATICS, THE UNIVERSITY OF AUCKLAND, 2025.

Abstract

The rise of quantum algorithms capable of breaking widely deployed public-key schemes has spurred the search for post-quantum alternatives. Lattice-based and code-based cryptography are leading candidates, each with strong theoretical guarantees but different trade-offs. This thesis engages with both paradigms through two complementary directions: the study of a lattice-code scheme and the analysis of new algorithmic problems with cryptographic applications.

The first part focuses on the scheme of Li, Ling, Xing, and Yeo (LLXY), which introduces lattices supporting efficient Bounded Distance Decoding. We analyse the structure of these polynomial lattices, reformulate the construction as a Key Encapsulation Mechanism (KEM), and propose refinements that yield compact ciphertexts while ensuring security against adaptive Chosen Ciphertext Attacks. A comprehensive cryptanalytic evaluation examines resistance to state-of-the-art lattice and decoding attacks, provides conservative parameter choices, and highlights both the strengths and open challenges of this approach.

The second part investigates algorithmic problems inspired by the generalised birthday framework for $\{-1, 1\}^m$ vectors. We develop new Wagner-style k-list algorithms, analyse their complexity, and demonstrate applications to a new version of the Inhomogeneous Short Integer Solution (ISIS) assumption underlying a recent blind signature scheme. Our results reveal new combinatorial attacks, clarify the hardness of this assumption, and extend the framework to structured sums with arbitrary targets, supported by theoretical and experimental analysis.

Together, these contributions shed light on less studied directions of post-quantum cryptography and highlight the need for further scrutiny of both new constructions and their underlying hardness assumptions.

Acknowledgements

This journey would not have been possible without the generous support and help of many individuals.

First and foremost, I am deeply indebted to my supervisor, Steven Galbraith, for his outstanding guidance, supervision, and unwavering patience throughout my PhD. I am profoundly grateful for the opportunity to be your student and for the generosity with which you have shared your expertise, time, and attention over the past years.

I am also sincerely grateful to my co-supervisor Jurij Volcic for his insightful feedback and guidance during the writing of this thesis, and to Jeroen Schillewaert, who served as my co-supervisor in the early stages of my PhD and generously shared his time and expertise.

Above all, no one has been more important in this pursuit than my family, whose love, and encouragement have given me the strength to pursue my dreams. My heartfelt thanks go to my husband and my parents for their constant love and support, which have kept me motivated and confident. Finally, I would like to thank all my colleagues for their valuable feedback and for the friendship that made these years so rewarding.

Table of Contents

List of Figures

List of Tables

List of Algorithms

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

School of Graduate Studies
AskAuckland Central
Alfred Nathan House
The University of Auckland
Tel: +64 9 373 7599 ext 81321
Email: postgradinfo@auckland.ac.nz

Co-Authorship Form

This form is to accompany the submission of any PhD that contains published or unpublished co-authored work. **Please include one copy of this form for each co-authored work**. Completed forms should be included in all copies of your thesis submitted for examination and library deposit (including digital deposit), following your thesis Acknowledgements. Co-authored works may be included in a thesis if the candidate has written all or the majority of the text and had their contribution confirmed by all co-authors as not less than 65%.

Please indicate the chapter/section/pages of this thesis that are extracted from a co-authored work and give the title and publication details or details of submission of the co-authored work. Thesis Chapters 3,4, and 5 Publication detalis: P. Athukorala and S. D. Galbraith, "Breaking and Improving a Lattice-Code-Based Cryptosystem by Li, Ling, Xing, and Yeo," in IEEE Transactions on Information Theory, vol. 71, no. 9, pp. 6857-6869, Sept. 2025, doi: 10.1109/TIT.2025.3573912.	
Nature of contribution by PhD candidate	Wrote the paper, proofs and computations.
Extent of contribution by PhD candidate (%)	70%

CO-AUTHORS

Name	Nature of Contribution
Steven Galbraith	Supervision, Ideas, Wrote part of the paper, Feedback/review

Certification by Co-Authors

The undersigned hereby certify that:

- the above statement correctly reflects the nature and extent of the PhD candidate's contribution to this work, and the nature of the contribution of each of the co-authors; and
- that the candidate wrote all or the majority of the text.

Name	Signature	Date
Steven Galbraith		29/9/2025

Chapter 1

Introduction

The design and analysis of cryptographic schemes in the post-quantum era has emerged as a central challenge in modern cryptography. With the advent of quantum algorithms capable of breaking widely deployed public-key primitives, there is an urgent need to investigate alternative hardness assumptions and novel constructions that can withstand quantum adversaries, as highlighted by ongoing post-quantum standardisation efforts through the National Institute of Standards and Technology (NIST) competition. Among the most promising directions are *lattice-based cryptography*, which relies on the hardness of lattice problems such as Learning With Errors (LWE) and Short Integer Solution (SIS), and *code-based cryptography*, rooted in the difficulty of decoding random linear codes. Both paradigms offer strong theoretical foundations, but each comes with its own trade-offs in terms of efficiency, structure, and security. In our work, we engage with both these directions.

This thesis is structured in two main parts. The first half (Chapters 3-5) focuses on a recently proposed scheme by Li, Ling, Xing, and Yeo (LLXY), which aims to bridge lattice-based and code-based cryptography through a novel family of lattices that admits an efficient Bounded Distance Decoding (BDD) algorithm. We provide a detailed analysis of this lattice family together with a cryptanalytic study of the construction by Li et al. [LLXY20]. We propose further improvements and carry out a comprehensive security evaluation. The second half (Chapters 6-8) explores a new class of algorithmic problems involving the generalised birthday problem for $\{-1, 1\}^m$ vectors. These problems arise naturally in the analysis of a recent variant of the Inhomogeneous Short Integer Solution (ISIS) problem by Baldimtsi et al. [BCGY24]. We propose new combinatorial algorithms for this problem, investigate their complexity, and apply them to challenge the security of a recently introduced blind signature scheme.

The main contributions of this thesis can be summarised as follows.

1.1 Contributions of the First Half

The first part of the thesis is based on the publication [AG25].

In **Chapter 3**, we begin by examining the polynomial lattices introduced in the LLXY scheme [LLXY20]. We show that while the construction is elegant in its attempt to interpolate between lattice-based and code-based approaches, the original scheme is not secure against adaptive Chosen Ciphertext Attacks (CCA). Our analysis introduces a new structural formulation of polynomial lattices, which not only deepens theoretical understanding but also opens potential avenues for further cryptanalysis.

Chapter 4 redefines the LLXY scheme as a Key Encapsulation Mechanism (KEM) using Niederreiter's framework [Nie86], thereby achieving more compact ciphertexts while maintaining standard security levels. We show how to compute the parity-check matrix for codes over the finite ring $\mathbb{Z}_{q-1}$ directly, without first computing a generator matrix, and explain decoding for ternary errors (i.e., errors in $\{-1, 0, 1\}^n$) while avoiding decapsulation failures. We further demonstrate how to achieve Chosen Ciphertext Attack security by applying standard techniques.

In **Chapter 5**, we present a full-fledged security analysis of the LLXY KEM, assessing its resistance to the state-of-the-art lattice-based attacks and decoding attacks. We provide conservative parameter choices and highlight the scheme's flexibility in extending error distributions beyond the ternary setting. On a practical note, we compute the public key size and secret key size and provide a comparison against the cryptographic counterparts like Kyber [BDK$^+$18], McEliec [BCL$^+$17], BIKE [ABB$^+$22], and HQC [MAB$^+$18]. While our study demonstrates the scheme's strengths, we also emphasise the need for continued cryptanalytic investigation of this special lattice family to evaluate its potential for use in cryptographic schemes.

1.2 Contributions of the Second Half

In **Chapter 6**, we shift focus to a new computational problem: finding vectors from $\{-1, 1\}^m$ that add up to the zero vector across multiple lists. This can be seen as a variant of the generalised birthday problem introduced by Wagner [Wag02]. We analyse its complexity and introduce a Wagner-like k-tree algorithm to obtain better running time. To the best of our knowledge, this is the first systematic study of the problem. Beyond its

algorithmic interest, we also show that it has direct cryptographic relevance, as it arises naturally in the security analysis of a new hardness assumption.

Building on this, **Chapter 7** applies the new k-tree algorithm to solve the randomised one-more ISIS (rOM-ISIS) assumption, a recently proposed hardness assumption underlying a blind signature scheme [BCGY24]. We propose a combinatorial attack that exploits the additive structure of rOM-ISIS and show that additional valid solutions can be generated with complexity exponential in the dimension. Consequently, while the attack is not feasible for a probabilistic polynomial-time adversary and the blind signature scheme remains secure under its current parameter sets, our results indicate that the underlying assumption is weaker than originally claimed.

Finally, in **Chapter 8**, we generalise the structured sum problem introduced in Chapter 6 by allowing a fixed target vector instead of the zero vector. We adapt and analyse a Wagner-style k-list algorithm for this setting, providing both theoretical insights and experimental validation. We observe that the complexity of the algorithm depends on the specific target vector. For completeness of the complexity analysis, we give an overall average running time by averaging over many possible target vectors.

1.3 Overall Outlook

Taken together, the two halves of the thesis contribute to advancing our understanding of both *cryptographic design* and *cryptanalysis* in the post-quantum setting.

The study of the LLXY KEM highlights both the potential and the challenges of hybrid constructions that combine ideas from lattice- and code-based cryptography. Through promising results on the LLXY KEM, we encourage further research on the joint field of lattice- and code-based post-quantum cryptography. The exploration of variants of the generalised birthday problem and their associated algorithms open new directions for analysing hardness assumptions derived from lattice problems, with implications for the design of secure and efficient post-quantum primitives.

This dual focus reflects the central theme of the thesis: that progress in post-quantum cryptography depends on the continual interplay between *new constructions* and their *cryptanalytic evaluation*.

Chapter 2

Background

This chapter provides the background necessary for this work. This includes the precise mathematical definitions and cryptographic terminologies we need throughout the thesis.

2.1 Notation

We denote the set of integers, rationals, reals, and non-negative integers by $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{N}$, respectively. For $q \in \mathbb{N}$, $\mathbb{Z}_q$ denotes the ring of integers modulo q, and a finite field of order q is denoted by $\mathbb{F}_q$. Let $\mathbb{Z}_q^n$ denote the set of n-dimensional vectors with entries in $\mathbb{Z}_q$. Let $\oplus$ denote the bitwise XOR operation.

Vectors are written in bold lowercase (e.g., $\mathbf{x}$). We write $\|\mathbf{x}\|_1$ for the ℓ_1-norm of a vector $\mathbf{x}$, and $\|\mathbf{x}\| = \|\mathbf{x}\|_2$ denotes the Euclidean norm. Concatenation of two vectors is denoted by $(\mathbf{x} \parallel \mathbf{y})$. We denote matrices with boldface capital letters. If $\mathbf{M}$ is a matrix, we usually adopt the compact notation $\mathbf{M}_{i,j}$ to indicate the element in the i^{th} row and j^{th} column, and denote with $\mathbf{M}^\top$ its transpose. We write $\mathbf{I}_m$ for the $m \times m$ identity matrix.

Choosing an element x at random from a set or distribution $\mathcal{D}$ is denoted by $x \leftarrow\!\$\ \mathcal{D}$. We use $\Pr[E]$ to denote the probability that an event E occurs. The expected value of a random variable x is often denoted by $\mathbb{E}[x]$.

Birthday problem In probability theory, the birthday problem asks for the probability that, in a set of n randomly chosen people, at least two will share the same birthday. In cryptography, we use this as a probabilistic tool that is used to determine the expected number of elements that need to be drawn from a given set until a collision is found (Section 14.1 [Gal12]).

Definition 2.1 (Birthday Problem)**.** *Given two lists L_1, L_2 of elements drawn uniformly and independently at random from $\{0,1\}^n$, find $\mathbf{x}_1 \in L_1$ and $\mathbf{x}_2 \in L_2$ such that $\mathbf{x}_1 \oplus \mathbf{x}_2 = \mathbf{0}$.*

Since $\mathbf{x}_1 \oplus \mathbf{x}_2 = \mathbf{0}$ if and only if $\mathbf{x}_1 = \mathbf{x}_2$, the task reduces to finding a collision between the two lists. A straightforward strategy is to sort both lists and check for equal elements, or equivalently to compare all pairs. However, these approaches are computationally inefficient, and motivate the study of more effective collision-finding methods that exploit probabilistic arguments rather than exhaustive comparison.

In Part II of the thesis we work extensively with algorithms that has foundation in the birthday problem. The following theorem provides a probabilistic estimate for the list sizes required to expect such a collision.

Theorem 2.1 [Gal12]. *Let S be a set of N elements and let samples be drawn independently and identically distributed (i.i.d.) uniformly from S. Then the expected number of samples to be taken before some element is sampled twice is less than $\sqrt{\pi N/2} + 2 \approx 1.253\sqrt{N}$.*

2.2 Asymptotic Complexity

Asymptotic complexity is the key to comparing algorithms. The time complexity of an algorithm summarises how the execution time of an algorithm grows with the input size. The space complexity similarly summarises how the amount of memory an algorithm requires grows with the input size. Both these complexity measures ignore constant factors, because those depend on machine details such as instruction set or clock rate. Complexity measures instead focus on asymptotic growth, which is independent of the particular machine and thus permit comparison of different algorithms without regard to particular hardware. For sufficiently large inputs, asymptotic effects will dominate any constant factor advantage.

In this thesis, we use standard asymptotic notations to describe the growth of functions. Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be functions.

- **Big-O notation:** We write $f(n) \in \mathcal{O}(g(n))$ if there exist constants $c > 0$ and $n_0 \in \mathbb{N}$ such that
$$f(n) \leq c \cdot g(n) \quad \text{for all } n \geq n_0.$$
This gives an *asymptotic upper bound* on f.

- **Big-Ω notation:** We write $f(n) \in \Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \in \mathbb{N}$ such that
$$f(n) \geq c \cdot g(n) \quad \text{for all } n \geq n_0.$$
This gives an *asymptotic lower bound* on f.

- **Big-Θ notation:** We write $f(n) \in \Theta(g(n))$ if there exist constants $c_1, c_2 > 0$ and $n_0 \in \mathbb{N}$ such that

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \text{for all } n \geq n_0.$$

  This means f is bounded above and below by constant multiples of g for sufficiently large n.

Asymptotic complexity often simplifies the description of an algorithm's performance by focusing on the dominant growth rate, potentially hiding polynomial or logarithmic factors (unless otherwise specified). For instance, an algorithm with a complexity of $\mathcal{O}(n + \log n)$ simplifies to $\mathcal{O}(n)$ because for large values of n, the linear term (n) dominates the logarithmic term $(\log n)$.

2.3 Cryptographic primitives

This section discusses the basic cryptographic schemes that will be used throughout the thesis. In accordance with standard practice in modern cryptography, we provide precise mathematical definitions of the schemes and their corresponding security properties [KL07, Gal12].

2.3.1 Public-Key Encryption scheme

Let 1^λ be a security parameter. A Public-Key Encryption (PKE) scheme is a triple of probabilistic, polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ defined over three spaces as given in Figure 2.1. Let $\mathcal{K}$ be a key space, where the public key space is denoted by $\mathcal{K}_{\text{publ}}$ and the private key space by $\mathcal{K}_{\text{priv}}$. The set of messages to be encrypted, or *plaintext space*, is given by $\mathcal{P}$ and depends on the public key. $\mathcal{C}$ denotes the set of the messages transmitted over the channel, or *ciphertext space*.

It is required that, for all $(pk, sk) \leftarrow\!\!\$\ \mathsf{KeyGen}$ and all messages $\mathsf{m} \in \mathcal{P}$, we have $\mathsf{Dec}(\mathsf{Enc}(\mathsf{m}, pk), sk) = \mathsf{m}$ except with negligible probability over the randomness of KeyGen and Enc. This property is referred to as (perfect) correctness.

Security of Encryption

To analyse the security of a public-key encryption scheme, we formalise the following security properties. A basic requirement is *One-Way encryption*, which ensures that given a ciphertext and the public key, it is infeasible for an adversary to recover the plaintext.

$\mathsf{KeyGen}(1^\lambda)$: A probabilistic key generation algorithm that takes as input a security parameter 1^λ and outputs a public key $pk \in \mathcal{K}_{\text{publ}}$ and a private key $sk \in \mathcal{K}_{\text{priv}}$. Assume that pk and sk each have length at least λ, and λ can be determined from pk and sk.

$\mathsf{Enc}(pk, \mathsf{m})$: A (possibly probabilistic) encryption algorithm that receives as input a public key $pk \in \mathcal{K}_{\text{publ}}$ and a plaintext $\mathsf{m} \in \mathcal{P}$ and returns a ciphertext $\mathsf{c} \in \mathcal{C}$. Denote this by $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{m}, pk)$.

$\mathsf{Dec}(sk, \mathsf{c})$: A deterministic decryption algorithm that receives as input a private key $sk \in \mathcal{K}_{\text{priv}}$ and a ciphertext $\mathsf{c} \in \mathcal{C}$ and outputs either a plaintext $\mathsf{m} \in \mathcal{P}$ or the failure symbol $\perp$. Write this as $\mathsf{m} := \mathsf{Dec}(\mathsf{c}, sk)$.

Figure 2.1: Public-Key Encryption scheme

We define the notion of one-wayness under chosen-plaintext attacks (OW-CPA) as follows.

Definition 2.2 (OW-CPA)**.** *A deterministic public-key encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be one-way under chosen-plaintext attack (OW-CPA) if for every probabilistic polynomial-time adversary* $\mathcal{A}$*,*

$$\Pr[\mathcal{A}(pk, \mathsf{Enc}(pk, \mathsf{m})) = \mathsf{m}] \leq \mathit{negl}(\lambda),$$

where $(pk, sk) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$ *and* $\mathsf{m} \leftarrow\!\!\$\ \mathcal{P}$*.*

OW-CPA captures the minimal requirement that a ciphertext does not reveal enough information to efficiently recover the underlying plaintext. *Semantic security* ensures that an adversary learns no information at all about a message from its ciphertext, apart from possibly the length of the message. The *Indistinguishability* (IND) property requires that an adversary cannot distinguish between encryptions of two chosen plaintexts, chosen by the adversary, of the same length. These notions form the basis for modern security definitions.

Indistinguishability can be achieved in various attack models. We present here two of the most widely studied: chosen-plaintext attacks (CPA) and adaptive chosen-ciphertext attacks (CCA2).

Definition 2.3 (IND-CPA)**.** *Run* $\mathsf{KeyGen}(1^\lambda)$ *to obtain keys* (pk, sk)*. An adversary* $\mathcal{A}$ *for indistinguishability under chosen-plaintext attacks is given* pk *and in the challenge phase,* $\mathcal{A}$ *outputs two messages* $\mathsf{m}_0, \mathsf{m}_1 \in \mathcal{P}$ *of equal length. A uniform bit* $b \in \{0, 1\}$ *is chosen and the challenger returns* $\mathsf{c}^* \leftarrow \mathsf{Enc}(\mathsf{m}_b, pk)$*. The adversary outputs a guess* b^**. We define*

the advantage as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}(\lambda) = \left|\Pr[b^* = b] - \tfrac{1}{2}\right|.$$

A scheme is IND-CPA secure if for all probabilistic polynomial-time adversaries this advantage is negligible in the security parameter.

Definition 2.4 (IND-CCA2)**.** *Run* $\mathsf{KeyGen}(1^\lambda)$ *to obtain keys* (pk, sk)*. An adversary* $\mathcal{A}$ *for indistinguishability under adaptive chosen-ciphertext attacks is given* pk *and has oracle access to the decryption oracle* $\mathsf{Dec}_{sk}(\cdot)$*, with the restriction that the challenge ciphertext* c^* *cannot be queried to the decryption oracle. In the challenge phase,* $\mathcal{A}$ *outputs two messages* $\mathsf{m}_0, \mathsf{m}_1 \in \mathcal{P}$ *of the same length. A uniform bit* $b \in \{0, 1\}$ *is chosen and the challenger returns* $\mathsf{c}^* \leftarrow \mathsf{Enc}(\mathsf{m}_b, pk)$*. The adversary may continue to query the decryption oracle on any* $\mathsf{c} \neq \mathsf{c}^*$ *before outputting a guess* b^**. The advantage is defined as*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CCA2}}(\lambda) = \left|\Pr[b^* = b] - \tfrac{1}{2}\right|.$$

A scheme is IND-CCA2 secure if for all probabilistic polynomial-time adversaries, this advantage is negligible in the security parameter.

2.3.2 Key Encapsulation Mechanism

A *Key Encapsulation Mechanism* (KEM) [Den03] is a cryptographic primitive that allows anyone in possession of some party's public key to securely transmit a key to that party. A KEM can be viewed as a key-exchange protocol in which only a single message is transmitted; the main application is in combination with symmetric encryption to achieve public-key encryption of messages of arbitrary length.

A KEM is a triple of probabilistic polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap})$ and a corresponding key space $\mathcal{K}$. The *Key Generation* algorithm generates a pair of public and private keys. In place of encryption algorithm in PKE, we now have an *Encapsulation* algorithm that takes only a public key as input and no message and outputs a ciphertext along with a key. A *Decapsulation* algorithm is run to recover the key from the ciphertext using the private key.

It is required that with all but negligible probability over the randomness of $\mathsf{KeyGen}(1^\lambda)$ and $\mathsf{Encap}(pk)$, if $\mathsf{Encap}(pk)$ outputs (c, k), then $\mathsf{Decap}(\mathsf{c}, sk)$ outputs k.

For $\epsilon \geq 0$, a KEM is ϵ-correct if for all $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $(\mathsf{c}, k) \leftarrow \mathsf{Encap}(pk)$, it holds that

$$\Pr[\mathsf{Decap}(sk, \mathsf{c}) \neq k] \leq \epsilon.$$

$\mathsf{KeyGen}(1^\lambda)$: A probabilistic key generation algorithm that takes as input a security parameter 1^λ and outputs a public/private key pair (pk, sk). Assume that pk and sk each have length at least λ, and λ can be determined from pk.

$\mathsf{Encap}(pk)$: A probabilistic encapsulation algorithm that takes as input a public key pk (which implicitly defines λ) and outputs a ciphertext c and a key $k \in \mathcal{K}$. Write this as $(\mathsf{c}, k) \leftarrow \mathsf{Encap}(pk)$

$\mathsf{Decap}(\mathsf{c}, sk)$: A deterministic decapsulation algorithm that takes as input a secret key sk and a ciphertext c and returns a key $k \in \mathcal{K}$ or $\perp$ denoting failure. Write this as $k := \mathsf{Decap}(\mathsf{c}, sk)$.

Figure 2.2: Key Encapsulation Mechanism.

A KEM is correct if $\epsilon = 0$. The security of a KEM ensures that an adversary without the private key cannot distinguish the encapsulated key from a random value. Therefore, the security of KEMs is defined in terms of the indistinguishability of the session key against chosen-plaintext (IND-CPA) and chosen-ciphertext (IND-CCA) adversaries.

In the notion of CPA-security for PKEs, it requires that the attacker is unable to distinguish whether ciphertext c an encryption of some message m_0 or some other message m_1. With a KEM there is no message, and we require instead that the encapsulated key k is indistinguishable from a uniform key that is independent of the ciphertext c.

Definition 2.5 (IND-CPA for KEM)**.** *Run* $\mathsf{KeyGen}(1^\lambda)$ *to obtain keys* (pk, sk) *and then* $\mathsf{Encap}(pk)$ *to generate* (c, k)*. A uniform bit* $b \in \{0, 1\}$ *is chosen and if* $b = 0$*, set* $\hat{k} = k$*. If* $b = 1$*, then chose a uniform* $\hat{k} \in \mathcal{K}$*.*

An adversary $\mathcal{A}$ *for indistinguishability under chosen-plaintext attacks is given* $(pk, \mathsf{c}, \hat{k})$ *and in the challenge phase,* $\mathcal{A}$ *outputs a bit* b^**. We define the advantage as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},KEM}(\lambda) = \left|\Pr[b^* = b] - \tfrac{1}{2}\right|.$$

A KEM is IND-CPA secure if for all probabilistic polynomial-time adversaries this advantage is negligible in the security parameter.

Definition 2.6 (IND-CCA for KEM)**.** *Run* $\mathsf{KeyGen}(1^\lambda)$ *to obtain keys* (pk, sk) *and then* $\mathsf{Encap}(pk)$ *to generate* (c, k)*. A uniform bit* $b \in \{0, 1\}$ *is chosen and if* $b = 0$*, set* $\hat{k} = k$*. If* $b = 1$*, then chose a uniform* $\hat{k} \in \mathcal{K}$*. An adversary* $\mathcal{A}$ *for indistinguishability under adaptive chosen-ciphertext attacks is given* $(pk, \mathsf{c}^*, \hat{k})$ *and has oracle access to the decapsulation oracle* $\mathsf{Decap}_{sk}(\cdot)$*, with the restriction that the challenge ciphertext* c^* *cannot be queried to*

the decap oracle. In the challenge phase, the adversary may continue to query the decap oracle on any $\mathsf{c} \neq \mathsf{c}^*$ *before outputting a guess* b^**. The advantage is defined as*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},KEM}(\lambda) = \left|\Pr[b^* = b] - \tfrac{1}{2}\right|.$$

A KEM is IND-CCA2 secure if for all probabilistic polynomia-time adversaries, this advantage is negligible in the security parameter.

KEMs are widely used in hybrid encryption schemes, where they provide secure key exchange for symmetric encryption.

2.3.3 Signature schemes

A signature scheme is defined, analogously to encryption, by message, signature and key spaces depending on a security parameter λ. There is a KeyGen algorithm and algorithms Sign and Verify. Let $\mathcal{K}_{\mathsf{Sign}}$ be the signing key space and $\mathcal{K}_{\mathsf{Verify}}$ be the verification key space. The set of documents to be signed, or message space, is $\mathcal{P}$ and the set of the signatures to be transmitted along with the messages, or signature space, is Σ.

$\mathsf{KeyGen}(1^\lambda)$: A probabilistic key generation algorithm that takes as input a security parameter 1^λ and outputs a signing key $sk \in \mathcal{K}_{\mathsf{Sign}}$ and a verification key $vk \in \mathcal{K}_{\mathsf{Verify}}$.

$\mathsf{Sign}(\mathsf{m}, sk)$: A (possibly probabilistic) signing algorithm that receives as input a signing key $sk \in \mathcal{K}_{\mathsf{Sign}}$ and a message $\mathsf{m} \in \mathcal{P}$ and returns a signature $\sigma \in \Sigma$.

$\mathsf{Verify}(\mathsf{m}, vk, \sigma)$: A deterministic decryption algorithm that receives as input a verification key $vk \in \mathcal{K}_{\mathsf{Verify}}$, a message $\mathsf{m} \in \mathcal{P}$ and a signature $\sigma \in \Sigma$ and outputs 1, if the signature is recognized as valid, or 0 otherwise.

Figure 2.3: Signature Scheme

We require that except with negligible probability over (sk, vk) output by $\mathsf{KeyGen}(1^\lambda)$, it holds that $\mathsf{Verify}(\mathsf{m}, \mathsf{Sign}(\mathsf{m}, sk), vk) = 1$ for every valid message $\mathsf{m} \in \mathcal{P}$.

The main attack goals against signature schemes are as follows. A *total break* occurs when an adversary recovers the private key corresponding to the given public key. In a *selective forgery* (also called target message forgery), the adversary is able to generate a valid signature under the given public key on a specific chosen message. Finally, in an *existential forgery*, the adversary produces at least one valid pair (m, σ), where m is any

message that has never been signed by the legitimate signer and σ is its corresponding signature under the given public key.

The formal definition of security of a signature scheme is as follows:

Definition 2.7. *Run* $\mathsf{KeyGen}(1^\lambda)$ *to obtain keys* (pk, sk). *An adversary* $\mathcal{A}$ *is given* pk *and access to an oracle* $\mathsf{Sign}_{sk}(\cdot)$. *The adversary then outputs* (m, σ). *Let* Q *denote the set of all quesries* $\mathcal{A}$ *made to its oracle.* $\mathcal{A}$ *succeeds if and only if*

1. $\mathsf{Verify}_{pk}(\mathsf{m}, \sigma) = 1$ *and*

2. $\mathsf{m} \notin Q$.

In this case the output of the experiment is defined to be 1.

A signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\Pr[\textit{Sig-forge}_{\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

Blind Signature scheme

The goal of a blind signature scheme [Cha83] is to allow a receiver to obtain a signature on a message of the receiver's choice without revealing this message to the signer. Typically, blind signatures are implemented as an interactive protocol between the signer (holding a secret key, sk) and the user (holding the message m to be signed and the signer's public verification key, vk). At the end of their interaction, the receiver should obtain a valid signature σ on m.

Formally, a blind signature scheme is a four-tuple consisting of two interactive Turing machines, the Signer and the User, (S, U), and two algorithms $(\mathsf{KeyGen}, \mathsf{Verify})$ [JLO97]. The complete definition of a blind signature scheme is given in Figure 2.4.

A blind digital signature scheme should satisfy two security properties: *blindness* (i.e., the signer does not obtain any information with respect to m) and *one-more unforgeability* (i.e., the receiver cannot create more valid signatures beyond the ones received after interacting with the signer).

- **Blindness:** Let $b \leftarrow\$ \{0, 1\}$. A PPT adversary $\mathcal{A}$ chooses two messages $\mathsf{m}_0, \mathsf{m}_1$ and interacts in parallel with two users, U_0 and U_1, who receive m_b and m_{1-b} respectively. If both users output valid signatures, $\mathcal{A}$ is given both signatures ordered

$\mathsf{KeyGen}(1^\lambda)$: A probabilistic key generation algorithm that takes as input a security parameter 1^λ and outputs a signing key $sk \in \mathcal{K}_{\mathsf{Sign}}$ and a verification key $vk \in \mathcal{K}_{\mathsf{Verify}}$.

$S(sk, vk), U(vk, \mathsf{m})$: S and U engage in an interactive protocol of some polynomial (in the security parameter) number of rounds. At the end of this protocol S outputs either "completed" or "not-completed" and U outputs either "fail" or a signature σ on m. The signing algorithm $\mathsf{Sign}(sk, \mathsf{m})$ is implicitly defined by this interaction; namely, $\mathsf{Sign}(sk, \mathsf{m})$ outputs the signature $\sigma(m)$ obtained by simulating the protocol between $S(sk, vk)$, and $U(vk, \mathsf{m})$.

$\mathsf{Verify}(vk, \mathsf{m}, \sigma(\mathsf{m}))$: A deterministic polynomial-time algorithm, which outputs "accept"/"reject" with the requirement that for any message m, and for all random choices of key generation algorithm, if both S and U follow the protocol then S always outputs "completed", and the output of U is always accepted by the verification algorithm.

Figure 2.4: Blind Signature Scheme

as $(\sigma(\mathsf{m}_0), \sigma(\mathsf{m}_1))$. $\mathcal{A}$ then outputs a guess $\hat{b}$. The scheme is blind if there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\Pr[\hat{b} = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

- **One-more Unforgeability:** The adversary $\mathcal{A}$ interacts adaptively with the signer and obtains at most ℓ valid signatures through the signing protocol. Eventually, $\mathcal{A}$ outputs $(\mathsf{m}_1, \sigma_1), \ldots, (\mathsf{m}_j, \sigma_j)$ such that

$$\mathsf{Verify}(pk, \mathsf{m}_i, \sigma_i) = 1 \quad \text{for all } 1 \leq i \leq j.$$

The scheme is one-more unforgeable if no PPT adversary can produce more valid signatures than the number of completed signing interactions, i.e., $j > \ell$ occurs only with negligible probability in the security parameter λ.

2.4 Lattices

An n-dimensional *lattice* $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^n$. Given linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$, the lattice generated by them is

$$\mathcal{L} = \left\{ \mathbf{x} \in \mathbb{R}^n \;\middle|\; \mathbf{x} = \sum_{i=1}^{d} z_i \mathbf{b}_i \text{ for } z_i \in \mathbb{Z} \right\}.$$

The $d \times n$ matrix $\mathbf{B}$ with rows $\mathbf{b}_1, \ldots, \mathbf{b}_d$ is called a *basis* of $\mathcal{L}$. The integer d is the *rank* of the lattice, which is the number of vectors in a basis of a lattice. A lattice is *full rank* if the rank of the lattice is equal to the dimension, i.e., $d = n$. For a point $\mathbf{t} \in \mathbb{R}^n$ and a lattice $\mathcal{L} \subset \mathbb{R}^n$, define the *distance* from $\mathbf{t}$ to the lattice as $\mathsf{dist}(\mathbf{t}, \mathcal{L}) = \min_{\mathbf{x} \in \mathcal{L}} \|\mathbf{t} - \mathbf{x}\|$. The (centred) fundamental parallelepiped of a lattice basis $\mathbf{B}$ is given by

$$\mathcal{P}(\mathbf{B}) = \left\{ \sum_{i=1}^{d} z_i \mathbf{b}_i \;\middle|\; -1/2 \leq z_i < 1/2 \quad \text{for all} \quad i \in \{1, \ldots, d\} \right\}.$$

The *volume* (or determinant) of a full-rank lattice $\mathcal{L}$ is $\mathrm{vol}(\mathcal{L}) = |\det(\mathbf{B})|$. In other words, it is the d-dimensional volume of the fundamental parallelepiped of a basis of $\mathcal{L}$.

The set of all vectors in $\mathbb{R}^n$ whose inner product with every vector in a full-rank lattice $\mathcal{L} \subset \mathbb{R}^n$ is an integer is called the *dual lattice*, denoted by $\mathcal{L}^*$.

$$\mathcal{L}^* = \{ \mathbf{y} \in \mathbb{R}^n \;:\; \langle \mathbf{y}, \mathbf{x} \rangle \in \mathbb{Z} \text{ for all } \mathbf{x} \in \mathcal{L} \}.$$

Gram-Schmidt Orthogonalization The Gram-Schmidt algorithm takes a vector space basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ as input and constructs an orthogonal basis $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$ recursively, beginning with $\mathbf{b}_1 = \mathbf{b}_1^*$ and then computing

$$\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle} \quad 1 < i \leq d, \;\; 1 \leq j < i \quad \text{and} \quad \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*.$$

Lattice reduction algorithms recursively modify the basis vectors to make them "size-reduced" and shorter. They continuously check the lengths of the Gram-Schmidt vectors to swap and reduce them until an acceptable quality is reached. A basis is considered "reduced" when the Gram-Schmidt vectors do not drop in length too drastically. This enables algorithms to find surprisingly short vectors in a lattice.

A special class of lattices we use are *q-ary lattices*, which are sublattices of $\mathbb{Z}^m$. For $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, define

$$\mathcal{L}_q^{\perp}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A}\mathbf{x}^{\top} = \mathbf{0} \pmod q\},$$

$$\mathcal{L}_q(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m \ : \ \mathbf{x} = \mathbf{s}\mathbf{A} \pmod q \text{ for some } \mathbf{s} \in \mathbb{Z}_q^n\}.$$

The lattice $\mathcal{L}_q(\mathbf{A})$ is the lattice generated by the rows of $\mathbf{A}$ modulo q and $\mathcal{L}_q^{\perp}(\mathbf{A})$ is the set of integer solutions in $\mathbb{Z}^m$ to the system of n linear equations modulo q defined by the rows of $\mathbf{A}$.

We write $\lambda_i(\mathcal{L})$ for *Minkowski's successive minima*, i.e., the radius of the smallest ball centred around zero containing i linearly independent lattice vectors. In particular, the length of the shortest non-zero vectors of a lattice $\mathcal{L}$ is denoted by $\lambda_1(\mathcal{L})$. The *Gaussian heuristic* estimates the length of the shortest non-zero vector in a full rank lattice $\mathcal{L}$ as

$$\lambda_1(\mathcal{L}) \approx \sqrt{\frac{n}{2\pi e}} \operatorname{vol}(\mathcal{L})^{1/n}. \tag{2.1}$$

2.4.1 Gaussian Functions and Discrete Gaussians

For $\mathbf{c} \in \mathbb{R}^n$ and $\sigma > 0$, the (spherical) Gaussian function is

$$\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}\|^2}{2\sigma^2}\right).$$

When $\mathbf{c} = \mathbf{0}$ we write $\rho_\sigma(\mathbf{x})$. An alternative parametrisation uses $\zeta = \sigma\sqrt{2\pi}$:

$$\rho_\zeta(\mathbf{x}) = \exp\left(-\pi\frac{\|\mathbf{x}\|^2}{\zeta^2}\right).$$

For a finite set $S \subset \mathbb{R}^n$, define $\rho_\zeta(S) = \sum_{\mathbf{x} \in S} \rho_\zeta(\mathbf{x})$. The *discrete Gaussian distribution* over a lattice $\mathcal{L}$ with parameter ζ is

$$\mathcal{D}_{\mathcal{L},\zeta}(\mathbf{x}) = \frac{\rho_\zeta(\mathbf{x})}{\rho_\zeta(\mathcal{L})}, \quad \mathbf{x} \in \mathcal{L}.$$

Micciancio and Regev [MR07] proposed a lattice quantity called the smoothing parameter, which quntifies the minimum amount of Gaussian noise that is needed to smooth out the discrete structure of a lattice. Above this threshold, a discrete Gaussian over the lattice begins to behave similarly to a continuous Gaussian distribution.

Definition 2.8. *For any n-dimensional lattice $\mathcal{L}$ and positive real $\epsilon > 0$, the smoothing parameter $\eta_\epsilon(\mathcal{L})$ is the smallest real $\zeta > 0$ such that $\rho_{1/\zeta}(\mathcal{L}^* \setminus \{\mathbf{0}\}) \leq \epsilon$.*

Lemma 2.1 [MR07, Lemma 4.3]. *For any n-dimensional lattice $\mathcal{L}$, vector $\mathbf{c} \in \mathbb{R}^n$, and reals $0 < \epsilon < 1$, $\zeta > 2\eta_\epsilon(\mathcal{L})$, where $\eta_\epsilon(\mathcal{L})$ is the smoothing parameter, we have*

$$\|\mathbb{E}_{\mathbf{x} \leftarrow \$ \mathcal{D}_{\mathcal{L},\mathbf{c},\zeta}}[\mathbf{x} - \mathbf{c}]\|^2 \leq \left(\frac{\epsilon}{1-\epsilon}\right)^2 \zeta^2 n$$

$$\mathbb{E}_{\mathbf{x} \leftarrow \$ \mathcal{D}_{\mathcal{L},\mathbf{c},\zeta}}[\|\mathbf{x} - \mathbf{c}\|^2] \leq \left(\frac{1}{2\pi} + \frac{\epsilon}{1-\epsilon}\right) \zeta^2 n$$

Lemma 2.2 [MR07, Lemma 4.4]. *For any n-dimensional lattice $\mathcal{L}$, vector $\mathbf{c} \in \mathbb{R}^n$, and reals $0 < \epsilon < 1$, $\zeta > \eta_\epsilon(\mathcal{L})$, we have*

$$\Pr_{\mathbf{x} \leftarrow \$ \mathcal{D}_{\mathcal{L},\mathbf{c},\zeta}}\left[\|\mathbf{x} - \mathbf{c}\| > \zeta\sqrt{n}\right] \leq \left(\frac{1+\epsilon}{1-\epsilon}\right) 2^{-n}.$$

2.4.2 Lattice Problems

Lattice-based cryptography relies on the assumed difficulty of solving computational problems on lattices. Below, we outline the key lattice problems that are central to this work.

Shortest Vector Problems. One of the most fundamental and extensively studied problems in lattice theory is the Shortest Vector Problem (SVP).

Definition 2.9 (SVP)**.** *Given a basis of $\mathcal{L}$, find a vector $\mathbf{x} \in \mathcal{L}$ with $\|\mathbf{x}\|_2 = \lambda_1(\mathcal{L})$.*

An important variant of SVP in lattice-based cryptography is the Unique Shortest Vector Problem (uSVP), which assumes a promise that the shortest non-zero vector is uniquely short.

Definition 2.10 (uSVP)**.** *Given $\mathcal{L}$ such that $\lambda_2(\mathcal{L}) \geq \gamma\lambda_1(\mathcal{L})$ for some $\gamma > 1$, find the shortest non-zero vector in $\mathcal{L}$.*

Closest Vector Problems. Another key problem is the Closest Vector Problem (CVP), which asks for a lattice vector nearest to a given target vector.

Definition 2.11 (CVP)**.** *Given a basis $\mathbf{B}$ for a lattice $\mathcal{L} \subset \mathbb{R}^n$ and a target point $\mathbf{t} \in \mathbb{R}^n$, find a lattice vector $\mathbf{x} \in \mathcal{L}$ with $\|\mathbf{t} - \mathbf{x}\|_2 = \mathrm{dist}(\mathbf{t}, \mathcal{L})$.*

A variant of the closest vector problem relevant in lattice-based cryptography is the Bounded Distance Decoding (BDD) problem.

Definition 2.12 (BDD)**.** *Given a basis of $\mathcal{L}$ and a target $\mathbf{t} \in \mathbb{R}^n$ satisfying $\operatorname{dist}(\mathbf{t}, \mathcal{L}) \leq \gamma\lambda_1(\mathcal{L})$ for some $\gamma < 1/2$, find $\mathbf{x} \in \mathcal{L}$ close to $\mathbf{t}$.*

Learning With Errors. Regev [Reg09] introduced the Learning With Errors (LWE) problem and proved a fundamental worst-case to average-case reduction for lattice problems. The average-case LWE problem is defined as follows.

Definition 2.13 (LWE)**.** *Let n, q be integers, χ be a probability distribution on $\mathbb{Z}$, and $\mathbf{s} \in \mathbb{Z}_q^n$ be a secret vector. The LWE distribution $\mathcal{A}_{\mathbf{s},\chi}$ outputs $(\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where, $\mathbf{a} \hookleftarrow\$ \mathbb{Z}_q^n$ uniformly, $e \hookleftarrow\$ \chi$, and $b = \langle \mathbf{a}, \mathbf{s} \rangle + e \pmod q$.*

Equivalently, one may view LWE in matrix form by collecting m independent samples. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be the matrix whose columns are the samples $\mathbf{a}_i$, and let $\mathbf{e} \in \mathbb{Z}_q^m$ be the error vector. Then the corresponding LWE instance is

$$\mathbf{b} = \mathbf{s}\mathbf{A} + \mathbf{e} \pmod q, \quad \text{where} \quad \mathbf{b} \in \mathbb{Z}_q^m.$$

The *search-LWE* problem is to recover $\mathbf{s} \in \mathbb{Z}_q^n$ from polynomially many samples drawn from $\mathcal{A}_{\mathbf{s},\chi}$. The *decision-LWE* problem is to distinguish polynomially many samples from the LWE distribution $\mathcal{A}_{\mathbf{s},\chi}$ from uniformly random samples in $\mathbb{Z}_q^n \times \mathbb{Z}_q$. Search-LWE can be seen as an average-case BDD problem on the family of q-ary lattices $\mathcal{L}_q(\mathbf{A}) \subseteq \mathbb{Z}^m$, where the target point is a noisy lattice vector $\mathbf{s}\mathbf{A} + \mathbf{e} \in \mathbb{Z}^m$.

In many instances, χ is taken to be a discrete Gaussian distribution over $\mathbb{Z}$ with mean zero and standard deviation $\alpha q/\sqrt{2\pi}$, for some $\alpha < 1$.

Short Integer Solution. Ajtai [Ajt96] introduced the well known Short Integer Solution (SIS) problem which asks to find a short solution to a linear algebra problem modulo q. Its inhomogeneous version was formalised later in [Mic07].

Definition 2.14 (SIS)**.** *Let q, n, m, β be functions of a security parameter λ. An instance of the $\mathsf{SIS}_{q,n,m,\beta}$ problem is a matrix $\mathbf{A} \hookleftarrow\$ \mathbb{Z}_q^{m \times n}$, and a solution to the problem is a non-zero integer vector $\mathbf{z} \in \mathbb{Z}^n$ such that $\|\mathbf{z}\|_2 \leq \beta$ and $\mathbf{A}\mathbf{z} = \mathbf{0} \pmod q$.*

Definition 2.15 (Inhomogeneous-SIS)**.** *Let q, n, m, β be functions of a security parameter λ. An instance of the $\mathsf{ISIS}_{q,n,m,\beta}$ problem is a matrix $\mathbf{A} \hookleftarrow\$ \mathbb{Z}_q^{m \times n}$ and a vector $\mathbf{y} \hookleftarrow\$ \mathbb{Z}_q^m$, and a solution to the problem is a vector $\mathbf{z} \in \mathbb{Z}^n$ such that $\|\mathbf{z}\|_2 \leq \beta$ and $\mathbf{A}\mathbf{z} = \mathbf{y} \pmod q$.*

2.4.3 Lattice Algorithms

In this section, we summarise the lattice algorithms and their behaviour that are relevant for this work.

Lattice Reduction

Lattice reduction (also known as lattice basis reduction or basis reduction) is the process of improving the quality of a lattice basis. It transforms a lattice basis into a more orthogonal one while preserving the lattice. Well-known lattice reduction algorithms include the LLL algorithm [LLL82] and the BKZ algorithm [SE94]. In this work we consider the BKZ-2.0 algorithm [CN11].

The BKZ algorithm is specified by a block size β, which is upper bounded by the rank of the lattice. The BKZ-β algorithm repeatedly calls an SVP oracle for finding (approximate) shortest non-zero vectors in projected lattices (also called local blocks) of dimension β. The basis obtained from BKZ-β reduction is called the BKZ-β reduced basis. In the Hermite-factor model, we assume that the BKZ-β reduced basis contains a vector of length,

$$\|\mathbf{b}_1\| = \delta^n \operatorname{vol}(\mathcal{L})^{1/n}, \quad \text{with} \quad \delta = \left(\frac{(\pi\beta)^{1/\beta} \cdot \beta}{2\pi e}\right)^{\frac{1}{2(\beta-1)}},$$

where δ is the root-Hermite factor [Che13]. We model the length of the vectors in the BKZ-β reduced basis using the Geometric Series Assumption (GSA)[Sch03, AGVW17] as follows.

Definition 2.16 (GSA)**.** *The norms of the Gram-Schmidt vectors after lattice reduction satisfy*

$$\|\mathbf{b}_i^*\| = \alpha^{i-1} \cdot \|\mathbf{b}_1\| \quad \textit{for some } 0 < \alpha < 1.$$

Combining the root-Hermite factor $\|\mathbf{b}_1\| = \delta^n \operatorname{vol}(\mathcal{L})^{1/n}$ with the GSA assumption, we get $\alpha = \delta^{-2n/(n-1)}$.

Babai's Nearest Plane Algorithm

Babai's Nearest plane algorithm (denoted by NP) is a decoding algorithm and is useful as a building block of several attacks and algorithms. For more details we refer to Babai's original work [Bab86] .

The inputs to the NP algorithm are a lattice basis $\mathbf{B} \subset \mathbb{Z}^d$ of full rank lattice and a target vector $\mathbf{t} \in \mathbb{R}^d$. The corresponding output is a vector $\mathbf{e} \in \mathbb{R}^d$ such that $\mathbf{t} - \mathbf{e} \in \mathcal{L}(\mathbf{B})$. We

denote the output by $\mathrm{NP}_{\mathbf{B}}(\mathbf{t}) = \mathbf{e}$. The output of the Nearest Plane algorithm satisfies the following condition, as shown in [Bab86].

Lemma 2.3. *Let $\mathbf{B} \subset \mathbb{Z}^d$ be a basis of a full rank lattice and $\mathbf{t} \in \mathbb{R}^d$ be a target vector. Then $NP_{\mathbf{B}}(\mathbf{t})$ is the unique vector $\mathbf{e} \in \mathcal{P}(\mathbf{B}^*)$ that satisfies $\mathbf{t} - \mathbf{e} \in \mathcal{L}(\mathbf{B})$, where $\mathbf{B}^*$ is the Gram-Schmidt basis of $\mathbf{B}$.*

2.4.4 Lattice Trapdoors

A *trapdoor* is a secret piece of information that enables efficient solutions to problems that are otherwise computationally hard. In the context of lattice based cryptography, lattices with trapdoors are statistically indistinguishable from randomly chosen lattices, yet they possess certain hidden structures that allow efficient solutions to hard lattice problems [MP12]. This concept underlies *trapdoor one-way functions*, which are efficiently computable in the forward direction but hard to invert without the trapdoor; given a trapdoor T, there exists an algorithm F^{-1} such that $F^{-1}(T, F(x)) = x$. Trapdoors are central to modern lattice-based cryptography because they enable secure and efficient constructions for public-key encryption, digital signatures, and other advanced primitives, while ensuring strong security against both classical and quantum adversaries.

We recall the trapdoor function framework of Gentry, Peikert, and Vaikuntanathan [GPV08], which provides collision-resistant hash functions together with an efficient trapdoor preimage sampler. The construction is given by three probabilistic polynomial-time algorithms $(\mathsf{TrapGen}, \mathsf{SampleDom}, \mathsf{SamplePre})$ that satisfy the following:

1. Generating a function with trapdoor: $\mathsf{TrapGen}(1^n)$ outputs (a, t), where a is the description of an efficiently computable function $f_a : D_n \to R_n$ (for some efficiently recognizable domain D_n and range R_n depending on n) and t is some trapdoor informaton for f_a.

   For the remainder, fix some $(a, t) \leftarrow \mathsf{TrapGen}(1^n)$.

2. Domain sampling and uniform output: $\mathsf{SampleDom}(1^n)$ samples an x from some (possibly non-uniform) distribution over D_n for which the distribution of $f_a(x)$ is uniform over R_n.

3. Preimage sampling with trapdoor: for every $y \in R_n$ $\mathsf{SamplePre}(t, y)$ samples from the consditional distribution of $x \leftarrow \mathsf{SampleDom}(1^n)$, given $f_a(x) = y$.

4. Preimage min-entropy: For every $y \in R_n$, the conditional min-entropy of $x \leftarrow \mathsf{SampleDom}(1^n)$, given $f_a(x) = y$ is at least $\omega(\log n)$.

5. Collision-resistance without trapdoor: for any probabilistic poly-time algorithm $\mathcal{A}$, the probability that $\mathcal{A}(1^n, a)$ outputs distinct $x, x' \in D_n$ such that $f_a(x) = f_a(x')$ is negligible, where the probability is taken over the choice of a and $\mathcal{A}$'s random coins.

In the second half of this thesis, we use the following definitions of TrapGen and SamplePre given in [BCGY24], which are adapted from the original framework in [GPV08].

Definition 2.17 (Trapdoor lattice sampler [BCGY24])**.** *For lattice parameters n, m, q with $n \geq \mathcal{O}(m \log q)$, a trapdoor lattice sampler consists of algorithms* TrapGen *and* SamplePre *with the following syntax:*

1. $\mathsf{TrapGen}(1^n, 1^m, q) \to (\mathbf{A}, T_{\mathbf{A}})$*: The lattice generation algorithm is a randomised algorithm that takes as input the matrix dimensions n, m, modulus q, and outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ together with a trapdoor $T_{\mathbf{A}}$.*

2. $\mathsf{SamplePre}(\mathbf{A}, T_{\mathbf{A}}, \mathbf{u}, \zeta) \to \mathbf{s}$*: The preimage sampling algorithm takes as input a matrix $\mathbf{A}$, trapdoor $T_{\mathbf{A}}$, a vector $\mathbf{u} \in \mathbb{Z}_q^m$, and a parameter $\zeta \in \mathbb{R}$ (which determines the length of the output vectors). It outputs a vector $\mathbf{s} \in \mathbb{Z}_q^n$ such that $\mathbf{As} = \mathbf{u}$ and $\|\mathbf{s}\| \leq \zeta\sqrt{n}$.*

These samplers are typically implemented using discrete Gaussian sampling over $\mathcal{L}_q^{\perp}(\mathbf{A})$ with appropiate centres.

2.5 Codes

For a prime or prime power q, a q-ary linear code over a field $\mathbb{F}_q$ is a linear subspace of $\mathbb{F}_q^n$. If k is the dimension of the subspace, then an $[n, k]$-linear code is said to have dimension k and length n. A *generator matrix* $\mathbf{G}$ for a linear code is a $k \times n$ matrix whose rows span the code. Given a generator matrix $\mathbf{G}$, the matrix $\mathbf{SG}$, where $\mathbf{S}$ is any invertible $k \times k$ matrix over $\mathbb{F}_q$, generates the same code. By permuting columns if necessary until the leftmost $k \times k$ submatrix is invertible, one can choose $\mathbf{S}$ so that the generator matrix is in the form $\mathbf{G} = \begin{pmatrix} \mathbf{I}_k & \mathbf{M} \end{pmatrix}$. This is called the *standard form* of the generator matrix.

The matrix $\mathbf{G}$ generates the code as a linear map: for each message $\mathbf{m} \in \mathbb{F}_q^k$, we obtain the corresponding codeword $\mathbf{mG}$. In systematic form, the first k positions of the codeword coincide with the message itself; these positions are called the *information symbols*.

Definition 2.18 (Dual code)**.** *Let C be an $[n, k]$-linear code over $\mathbb{F}_q$. The* dual code *of C is*

$$C^{\perp} = \{\mathbf{x} \in \mathbb{F}_q^n : \mathbf{x} \cdot \mathbf{y} = 0 \ \ \forall\, \mathbf{y} \in C\},$$

where $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i$ *is the standard inner product over* $\mathbb{F}_q$.

The dual code $C^\perp$ for a linear code C is an $[n, (n-k)]$-linear code of dimension $n-k$ and length n. If $\mathbf{G} = \begin{pmatrix} \mathbf{I}_k & \mathbf{M} \end{pmatrix}$ is a generator matrix in standard form of the code C, then $\mathbf{H} = \begin{pmatrix} -\mathbf{M}^\top & \mathbf{I}_{n-k} \end{pmatrix}$ is a generator matrix for $C^\perp$. The matrix $\mathbf{H}$ is a very important matrix for the code C itself. $\mathbf{H}$ is called the *parity check matrix* for the code, and it describes the code as follows:

$$\forall \mathbf{x} \in \mathbb{F}_q^n,\ \mathbf{x} \in C \iff \mathbf{x}\mathbf{H}^\top = 0.$$

In other words, if $\mathbf{G}$ is a generator matrix for a code C, then the parity check is an $(n-k) \times n$ matrix that satisfy the condition $\mathbf{G}\mathbf{H}^\top = \mathbf{0}$.

Remark 2.1. *In this thesis we use the convention that the parity-check matrix* $\mathbf{H} = \begin{pmatrix} -\mathbf{M} \\ \mathbf{I}_{n-k} \end{pmatrix}$, *is* $n \times (n-k)$ *with columns spanning* $C^\perp$, *so that* $\mathbf{G}\mathbf{H} = \mathbf{0}$ *holds. This is the transpose of the usual parity-check matrix definition, where rows span* $C^\perp$.

Codes are usually studied in the context of the Hamming metric.

Definition 2.19 (Hamming Metric)**.** *Let* q *be a prime power and* $\mathbb{F}_q$ *the finite field of order* q. *For* $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$, *the* Hamming distance *between* $\mathbf{x}$ *and* $\mathbf{y}$ *is defined as*

$$d_H(\mathbf{x}, \mathbf{y}) = |\{\, i \in \{1, \ldots, n\} \mid x_i \neq y_i \,\}|.$$

The Hamming weight *of* $\mathbf{x}$ *is* $\mathrm{wt}_H(\mathbf{x}) = d_H(\mathbf{x}, \mathbf{0})$, *i.e., the number of non-zero coordinates.*

Definition 2.20 (Minimum distance)**.** *Let* $C \subseteq \mathbb{F}_q^n$ *be a code. The* minimum Hamming distance *of* C *is*

$$d(C) = \min_{\substack{\mathbf{x},\mathbf{y} \in C \\ \mathbf{x} \neq \mathbf{y}}} d_H(\mathbf{x}, \mathbf{y}).$$

Alternative metrics, such as the Lee metric, are often used in other scenarios, for example codes over rings.

Definition 2.21 (Lee Metric)**.** *Let* $\mathbb{Z}_q$ *denote the ring of integers modulo* q. *For* $a \in \mathbb{Z}_q$, *the* Lee weight *of* a *is*

$$\mathrm{wt}_L(a) = \min(a,\ q - a),$$

where the integers are taken in $\{0, 1, \ldots, q-1\}$. *For a vector* $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_q^n$, *the* Lee weight *is*

$$\mathrm{wt}_L(\mathbf{x}) = \sum_{i=1}^{n} \mathrm{wt}_L(x_i).$$

The Lee distance *between* $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^n$ *is*

$$d_L(\mathbf{x}, \mathbf{y}) = \mathrm{wt}_L(\mathbf{x} - \mathbf{y}).$$

In the first half of the thesis we work with the ℓ_1 norm on $\mathbb{Z}^n$. To be precise, we write vectors $\mathbf{v} \in \mathbb{Z}_{q-1}^n$ as $\mathbf{v} = (v_1, \ldots, v_n)$, where $-\frac{(q-1)}{2} \leq v_i \leq \frac{q}{2} \quad \forall i$, and define the ℓ_1 norm as $\|\mathbf{v}\|_1 = \sum_{i=1}^n |v_i|$.

When working over the ring $\mathbb{Z}_q$, the corresponding distance measure in coding theory is the *Lee metric*. For $\mathbf{u}, \mathbf{v} \in \mathbb{Z}_q^n$, their Lee distance is defined as

$$d_L(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{n} \min\big(|u_i - v_i|,\, q - |u_i - v_i|\big),$$

where subtraction is performed in $\mathbb{Z}_q$ and each coordinate is interpreted as an integer in the range $[-(q-1)/2, q/2]$. In this representation, the Lee weight of $\mathbf{v}$ coincides with the ℓ_1 norm of its integer representative. Thus, the Lee metric can be seen as the ℓ_1 distance "wrapped" modulo q.

$(q-1)$-ary codes

Recall that every full-rank integer matrix can be transformed by unimodular row operations into a unique matrix in Hermite normal form (HNF).

Definition 2.22 (Hermite Normal Form)**.** *A matrix* $\mathbf{A} \in \mathbb{Z}^{m \times n}$ *has a (row) Hermite normal form* $\tilde{\mathbf{H}}$ *if there is a square unimodular matrix* $\mathbf{U}$ *such that* $\tilde{\mathbf{H}} = \mathbf{U}\mathbf{A}$ *and*

1. $\tilde{\mathbf{H}}$ *is upper triangular (i.e.,* $h_{i,j} = 0$ *for* $i > j$*),and any rows of zeros are located below any other row.*

2. *each pivot (first non-zero entry in a non-zero row) is always strictly to the right of the leading coefficient of the row above it; moreover, it is positive.*

3. *in each pivot column, all entries above the pivot satisfy* $0 \leq h_{i,j} < h_{j,j}$.

The HNF provides a canonical representation of an integer lattice. If q is prime and $\mathcal{L}$ is a is a q-ary lattice in $\mathbb{Z}^n$ with volume q^{n-k}, then a basis for $\mathcal{L}$ spans a dimension k code over $\mathbb{F}_q$. The generator matrix for the code is given by the first k rows of the Hermite normal form of a basis for the lattice.

A similar phenomenon can occur when q is not prime, as will be seen in the first half of the thesis.

For a finite ring R, a subset $C \subseteq R^n$ is called an R-submodule of R^n if it is closed under addition and scalar multiplication by elements of R. A linear code of length n over the ring R is an R-submodule of R^n. In this thesis we will work with codes over the finite ring $\mathbb{Z}_{q-1}$, which we call $(q-1)$-ary codes.

2.5.1 Syndrome Decoding

Given a linear code C with parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{n\times(n-k)}$, for any vector $\mathbf{x} \in \mathbb{F}_q^n$, the vector $\mathbf{xH} \in \mathbb{F}_q^{n-k}$ is called the *syndrome* of $\mathbf{x}$. The syndrome decoding method uses this to efficiently correct errors in received words.

Specifically, the code C partitions the space $\mathbb{F}_q^n$ into q^{n-k} distinct *cosets* of C. Each coset contains at least one vector of minimum Hamming weight. Any such vector is called a *coset leader*. In general, the coset leader is not unique; if multiple vectors have the same minimum Hamming weight, we select a unique representative using a fixed rule, such as lexicographic ordering.

Syndrome decoding works by pre-computing a table of syndromes and their corresponding coset leaders. When a received vector $\mathbf{z} = \mathbf{x} + \mathbf{e}$ is observed, where $\mathbf{x} \in C$ is the original codeword and $\mathbf{e}$ is an error vector, its syndrome satisfies:

$$\mathbf{zH} = \mathbf{xH} + \mathbf{eH} = \mathbf{0} + \mathbf{eH} = \mathbf{eH}.$$

If the weight of the error $\mathrm{wt}(\mathbf{e})$ is within the error-correcting radius of the code, i.e.,

$$\mathrm{wt}(\mathbf{e}) \leq \left\lfloor \frac{d-1}{2} \right\rfloor,$$

where d is the minimum distance of C, then $\mathbf{e}$ is the unique coset leader for the syndrome $\mathbf{zH}$. This enables the decoder to efficiently identify and correct the error by looking up the syndrome in the precomputed table.

Definition 2.23 (Syndrome Decoding Problem (SDP))**.** *Let q be a prime power, $\mathbf{H} \in \mathbb{F}_q^{n\times(n-k)}$ a parity-check matrix, and w a positive integer. Given a syndrome $\mathbf{s} \in \mathbb{F}_q^{(n-k)}$, the* Syndrome Decoding Problem *is to find a vector $\mathbf{e} \in \mathbb{F}_q^n$ such that $\mathbf{eH} = \mathbf{s}$ and $\mathrm{wt}(\mathbf{e}) \leq w$, where $\mathrm{wt}(\cdot)$ denotes the Hamming weight.*

2.5.2 Code based cryptography

Code-based cryptography studies cryptographic schemes which base their security on hard problem from coding theory. Classically, the underlying hard problem is the decoding of a

random linear code. This problem was shown to be NP-complete. McEliece proposed the first code-based encryption scheme in 1978 [McE78].

McEliece encryption scheme

The *McEliece scheme* is the first code-based public key encryption scheme, introduced by Robert McEliece. The public key of the McEliece scheme specifies a random binary Goppa code that has an efficient decoding algorithm.

Definition 2.24 (Classical Goppa codes)**.** *Fix a finite field $\mathbb{F}_{2^m}$ and consider a set of n distinct elements $\alpha_1, \ldots \alpha_n$ in $\mathbb{F}_{2^m}$. Fix an integer t, $2 \leq t \leq (2^m - 1)/m$, and an irreducible degree-t polynomial g.*

The Goppa code $\Gamma = \Gamma(\alpha_1, \ldots \alpha_n, g)$ consists of all elements $\mathbf{a} = (a_1, \ldots, a_n)$ in $\mathbb{F}_{2^m}$ satisfying

$$\sum_{i=1}^{n} \frac{a_i}{x - \alpha_i} \equiv 0 \pmod{g(x)}.$$

The dimension of the code Γ is at least $n - tm$ and the minimum distance is at least $2t + 1$.

A ciphertext of the scheme is a codeword plus random errors. The private key allows efficient decoding: extracting the codeword from the ciphertext, identifying and removing the errors.

The McEliece system was designed to be one-way (OW-CPA secure in the sense of Definition 2.2), meaning that an attacker cannot efficiently find the codeword from a ciphertext and public key, when the codeword is chosen randomly. Despite extensive cryptanalytic efforts over the years, the security level of the McEliece systems has remained remarkably stable.

The scheme can be easily generalised to codes over $\mathbb{F}_q$. Let $\mathcal{K}_{\text{publ}}$ be the set of $k \times n$ matrices over $\mathbb{F}_q$ and $\mathcal{K}_{\text{priv}}$ is the code description given by Γ. Plaintext is taken over the vector space $\mathbb{F}_q^k$ and the error vector over $\mathbb{F}_q^n$.

The McEliece system has prompted a tremendous amount of follow-up work. Some of this work improves efficiency while clearly preserving security: this includes a "dual" scheme proposed by Niederreiter.

Niederreiter cryptosystem

In contrast to McEliece, the *Niederreiter cryptosystem* [Nie86] is based on the dual of the error-correcting codes employed in the McEliece system, with its public key comprising a parity check matrix. We present a brief overview of the Niederreiter cryptosystem in Figure

KeyGen : Generate at random a polynomial $g \in \mathbb{F}_{q^m}[x]$ and elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{q^m}$, then build the Goppa code $\Gamma = \Gamma(\alpha_1, \ldots, \alpha_n, g)$ over $\mathbb{F}_q$ and its generator matrix $\hat{\mathbf{G}}$. Take $\mathbf{G}$ to be the standard form of $\hat{\mathbf{G}}$. Publish the public key $\mathbf{G} \in \mathcal{K}_{\text{publ}}$ and store the private key $\Gamma \in \mathcal{K}_{\text{priv}}$.

Enc : On input a public key $\mathbf{G} \in \mathcal{K}_{\text{publ}}$ and a plaintext $\mathbf{m} \in \mathbb{F}_q^k$, sample a random error vector $\mathbf{e}$ of weight w in $\mathbb{F}_q^n$ and return the ciphertext $\mathsf{c} = \mathbf{mG} + \mathbf{e} \in \mathbb{F}_q^n$.

Dec : On input the private key $\Gamma \in \mathcal{K}_{\text{priv}}$ and a ciphertext $\mathsf{c} \in \mathbb{F}_q^n$, first apply the decoding algorithm D_Γ to it. If the decoding succeeds, the first k information symbols of the decoded codeword reveal the plaintext. Otherwise, output $\perp$.

Figure 2.5: The McEliece encryption scheme

2.6. Note that here $\mathcal{K}_{\text{publ}}$ is the set of $(n-k) \times n$ matrices over $\mathbb{F}_q$ and $\mathcal{K}_{\text{priv}}$ is the code description Γ. The set of codewords of $\mathbb{F}_q^n$ with Hamming weight w is given by $\mathcal{P}$.

KeyGen : Generate at random a polynomial $g \in \mathbb{F}_{q^m}[x]$ and elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{q^m}$, then build the Goppa code $\Gamma = \Gamma(\alpha_1, \ldots, \alpha_n, g)$ over $\mathbb{F}_q$ and its parity-check matrix $\hat{\mathbf{H}}$. Take $\mathbf{H}$ to be the standard form of $\hat{\mathbf{H}}$. Publish the public key $\mathbf{H} \in \mathcal{K}_{\text{publ}}$ and store the private key $\Gamma \in \mathcal{K}_{\text{priv}}$.

Enc : On input a public key $\mathbf{H} \in \mathcal{K}_{\text{publ}}$ and a plaintext $\mathbf{e} \in \mathcal{P}$, with $\text{wt}(\mathbf{e}) = t$, compute the syndrome of $\mathbf{e}$, that is, $\mathbf{s} = \mathbf{He}^\top$ and return the ciphertext $\mathsf{c} = \mathbf{s} \in \mathbb{F}_q^{(n-k)}$.

Dec : On input the private key $\Gamma \in \mathcal{K}_{\text{priv}}$ and a ciphertext $\mathsf{c} \in \mathbb{F}_q^{(n-k)}$, first apply the decoding algorithm D_Γ to c. If the decoding succeeds, it returns the error vector $\mathbf{e}$, which is the plaintext. Otherwise, output $\perp$.

Figure 2.6: The Niederreiter cryptosystem

Classic McEliece [BCL+17] is a submission to the NIST Post-Quantum Cryptography standardisation competition as a candidate Key Encapsulation Mechanism(KEM). It is designed for IND-CCA2 security at a very high level, even against quantum computers. It is built conservatively from Niederreiter's version of McEliece's original public-key encryption scheme, instantiated with binary Goppa codes. Its main drawback lies in the large size of its public keys (hundreds of kilobytes or more), but it remains one of the most time-tested and robust candidates for post-quantum public-key encryption.

Part I

Breaking and Improving a Lattice-Code-based Cryptosystem.

Chapter 3

LLXY Scheme and Cryptanalysis

Contents

The interpolation of lattice-based and code-based cryptography has emerged as a promising path for advancing the design of post-quantum cryptographic schemes. In this context, Li, Ling, Xing, and Yeo (we will abbreviate as LLXY) [LLXY20] have made a notable contribution by proposing an encryption scheme that combines the structural

features of code-based encryption with trapdoors derived from lattice constructions over finite fields.

Specifically, the LLXY encryption scheme can be viewed as a variant of the classical McEliece encryption scheme, but adapted to $(q-1)$-ary linear codes. Unlike the original McEliece scheme, which relies on binary Goppa codes and hard decoding problems in the Hamming metric, the LLXY construction incorporates a decoding mechanism based on a lattice trapdoor tailored for the ℓ_1-norm. Security of the LLXY scheme is based on (but not proven to be equivalent to) a well-known hard lattice problem-Bounded Distance Decoding (BDD) problem (Definition 2.12). However, the trapdoor enables efficient decoding using polynomial factorisation, effectively blending code-based and lattice-based methodologies.

One of the key claims made by the authors is that their encryption scheme achieves security against Chosen Ciphertext Attacks (CCA), a strong adversarial model where the attacker is allowed to query a decryption oracle on arbitrary ciphertexts, excluding the challenge ciphertext. The main objective of this chapter is to conduct a comprehensive analysis of the LLXY encryption scheme, with a particular focus on its claimed CCA security. We demonstrate that, contrary to the authors' assertions, the scheme is vulnerable to a Chosen Ciphertext Attack that effectively breaks the proposed CCA protection.

Some potential advantages of the LLXY cryptosystem as a code-based cryptosystem are that it is not using the Hamming metric over binary fields (and so some of the best information set decoding attacks do not apply). In particular, the decoding algorithm can handle large Hamming weight error vectors (of small norm). This allows to securely use codes of much smaller length than is traditional in code-based cryptography. Some possible advantages as a lattice-based scheme are that decryption always works and does not require any reconciliation, and that the ciphertexts may potentially be smaller than other lattice schemes. Therefore, we believe that the scheme deserves further scrutiny.

3.1 Background & Motivation

The use of lattices with trapdoor functions has its roots in the early work of Chor and Rivest [CR84]. In 1988, they proposed a knapsack-based cryptosystem that leveraged discrete logarithms over finite field extensions. Although their scheme was later found to be insecure, their approach laid the groundwork for exploring the lattices with trapdoor functions in cryptographic applications.

Building on this line of research, Ducas and Pierrot [DP19] introduced a new family of lattices that admit efficient Bounded Distance Decoding (BDD) algorithms. Their construction is particularly notable for achieving decoding within a radius close to

Minkowski's bound both in the ℓ_1 and ℓ_2 norms which is typically hard for general lattices. One of the key features of their work is that both the construction of the lattice and the execution of the decoding algorithm have polynomial-time complexity, making the approach not only theoretically interesting but also potentially practical. This construction demonstrates that it is possible to embed trapdoors in dense lattices while retaining algorithmic efficiency for decoding problems, which are otherwise hard in worst-case scenarios.

We now provide a summary of the construction of this family of lattices, highlighting the main structural ideas and the decoding algorithm that accompanies them.

3.1.1 Discrete Logarithm Based Family of Lattices

This section describes the construction of a family of lattices based on discrete logarithms due to Ducas and Pierrot [DP19].

Fix a natural number n, and let m be a positive integer that is a prime power. We aim to construct a lattice of rank n embedded in $\mathbb{Z}^n$, whose structure is derived from the discrete logarithm problem in the multiplicative group $(\mathbb{Z}/m\mathbb{Z})^*$. The order of $(\mathbb{Z}/m\mathbb{Z})^*$ is given by $\phi(m)$ where ϕ is the Euler's totient function that counts the number of positive integers less than or equal to m that are coprime to m .

To define the lattice, choose n distinct primes $p_1, p_2, \ldots, p_n$ such that none of them divide m. To ensure their existence and manageable size, let B be a bound that depends on n, and pick the p_i such that $p_i \leq B$. Recall from the prime number theorem that the k^{th} prime is asymptotically equivalent to $k \log k$. Therefore, to find n small primes, it suffices to take $B \sim n \log n$.

Now consider the following group homomorphism:

$$\begin{aligned} \varphi : \mathbb{Z}^n &\to (\mathbb{Z}/m\mathbb{Z})^* \\ (x_1, \ldots, x_n) &\mapsto \prod_{i=1}^{n} p_i^{x_i} \pmod{m}. \end{aligned}$$

This map encodes integer vectors in $\mathbb{Z}^n$ as multiplicative combinations of the chosen primes modulo m. By the construction, the multiplicative inverse of $p_i \in (\mathbb{Z}/m\mathbb{Z})^*$ exists and denote by p_i^{-1}. The key object of interest is the kernel of this map:

$$\mathcal{L} := \ker(\varphi) = \left\{ (x_1, \ldots, x_n) \in \mathbb{Z}^n \;\middle|\; \prod_{i=1}^{n} p_i^{x_i} \equiv 1 \pmod{m} \right\}. \tag{3.1}$$

$\mathcal{L}$ is a subgroup of $\mathbb{Z}^n$ and is referred to as a *relation lattice*. To determine the rank of the lattice, observe that for each $i = 1, \ldots, n$ we have

$$\varphi(0, 0, \ldots, \phi(m), \ldots, 0) = p_i^{\phi(m)} \equiv 1 \pmod{m}.$$

Each of these points $(\phi(m), 0, \ldots, 0)$, $(0, \phi(m), 0, \ldots, 0), \ldots$ are in $\mathcal{L}$. These n points are linearly independent and generate a sublattice of $\mathcal{L}$ of rank n. Consequently, $\mathcal{L}$ also has rank n.

The volume or the determinant (denoted by $\mathrm{Vol}(\mathcal{L})$ and $\det(\mathcal{L})$ respectively) of the lattice is defined as the number of elements contained in a single fundamental parallelepiped of $\mathcal{L}$. Recall that $\mathrm{Im}(\varphi) \leq (\mathbb{Z}/m\mathbb{Z})^*$ since $(\mathbb{Z}/m\mathbb{Z})^*$ is finite abelian. By the first isomorphism theorem, it holds $\mathrm{Im}(\varphi) = \mathbb{Z}^n/\ker(\varphi)$ and so $|\mathrm{Im}(\varphi)| = |\mathbb{Z}^n/\mathcal{L}|$. Hence, the volume of the lattice is given by $|\mathrm{Im}(\varphi)|$ and thus bounded by $\phi(m)$. We say $\mathrm{Vol}(\mathcal{L}) \leq \phi(m)$ and the equality holds if and only if φ is surjective. In other words, if $\{p_1, \ldots, p_n\}$ generates $(\mathbb{Z}/m\mathbb{Z})^*$, then $\mathrm{Vol}(\mathcal{L}) = \phi(m)$.

Note that if $m = p^k$ for an odd prime p and $k > 0$, then $(\mathbb{Z}/m\mathbb{Z})^*$ is cyclic and therefore the generators of the group exists. If one of the primes p_i happens to be a generator of the multiplicative group $(\mathbb{Z}/m\mathbb{Z})^*$, then we are even able to compute a basis of the relation lattice. Without loss of generality, assume p_n is a generator of the multiplicative group $(\mathbb{Z}/m\mathbb{Z})^*$, then every element of the group can be expressed as a power of p_n. In other words, for each p_i, there exists a unique integer a_i such that $p_i \equiv p_n^{a_i} \pmod{m}$.

Then any vector in the kernel of φ (or equivalently in the relation lattice) satisfies:

$$\prod_{i=1}^{n} p_i^{x_i} \equiv 1 \pmod{m} \implies p_n^{\sum_{i=1}^{n-1} a_i x_i + x_n} \equiv 1 \pmod{m}.$$

This means the relation lattice $\mathcal{L}$ consists of all integer vectors $(x_1, \ldots, x_n) \in \mathbb{Z}^n$ satisfying the modular linear relation

$$\sum_{i=1}^{n-1} a_i x_i + x_n \equiv 0 \pmod{\phi(m)}.$$

This congruence allows us to explicitly describe a basis of $\mathcal{L}$ as the rows of the matrix

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -a_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & \phi(m) \end{pmatrix}.$$

This explicit description significantly simplifies working with the relation lattice in this special case where φ is surjective and p_n generates $(\mathbb{Z}/m\mathbb{Z})^*$.

Decoding Algorithm.

Ducas and Pierrot discuss several decoding radii for discrete and continuous errors. We will only look at the positive discrete bounded errors in this section.

Suppose we need to find a lattice point $\mathbf{v}$ when a lattice $\mathcal{L}$ and a target point $\mathbf{t}$ are given such that $\mathbf{t} = \mathbf{v} + \mathbf{e}$. The error vector $\mathbf{e}$ is a positive discrete bounded error such that the ℓ_1 norm is bounded. Say $\|\mathbf{e}\|_1 \leq r_{\mathbb{N}}$ for some bound $r_{\mathbb{N}}$. Now compute the following product modulo m.

$$\prod_{i=1}^{n} p_i^{t_i} = \prod_{i=1}^{n} p_i^{v_i+e_i} \equiv \prod_{i=1}^{n} p_i^{e_i} \pmod{m}.$$

The last equivalence holds because $\mathbf{v}$ is a lattice point. From $\|e\|_1 \leq r_{\mathbb{N}}$ and $p_i \leq B$ for any $i = 1, \ldots, n$, we get $\prod_{i=1}^{n} p_i^{e_i} \leq B^{r_{\mathbb{N}}}$. The authors claim that an efficient decoding can be ensured up to the ℓ_1 radius

$$r_{\mathbb{N}} = \frac{\ln m}{\ln B}.$$

In this case, $B^{r_{\mathbb{N}}} = m$. The factorisation is easy since the product $\prod_{i=1}^{n} p_i^{e_i}$ is less than m, and can be computed in $\mathbb{Z}$ not only modulo m. Then we factorise the integer which is smooth allowing us to recover $\mathbf{e}$.

A slight change in the decoding algorithm is needed when the error is discrete and bounded only, but not positive. We will discuss this case in a later section with its application to the LLXY setting.

Li et al. extended the concept of relation lattices by constructing them using polynomials over finite fields, leading to what they call as *polynomial lattices* [LLXY20, LLXY17]. In their earlier work [LLXY17], they addressed the Closest Vector Problem (CVP) in the context of polynomial lattices. Of particular interest to us is their later work [LLXY20], in which they applied these lattices to solve the Bounded Distance Decoding (BDD) problem.

3.2 The LLXY scheme

Li, Ling, Xing, and Yeo [LLXY20] proposed a variant of relation lattices and based on these lattices, they designed a new trapdoor function in which function inversion corresponds to solving the BDD problem efficiently. Based on this construction, they designed a public-key encryption scheme that is claimed to be CCA2 secure under suitable parameter choices.

This section provides a detailed exposition of the LLXY encryption scheme and the underlying polynomial lattice structure. We focus on a special case presented in [LLXY20], which the authors proposed for practical use and the related encryption scheme. Our goal is to describe the construction of the trapdoor function, and its integration into the encryption scheme, forming the basis for the cryptanalytic analysis in the following sections.

3.2.1 Setting & Construction of the LLXY Lattice

Let n and d be positive integers, and let q be a prime or prime power such that $q \geq n + d$. Consider the polynomial ring $\mathbb{F}_q[x]$ over the finite field $\mathbb{F}_q$. Let $c(x) \in \mathbb{F}_q[x]$ be a square-free polynomial of degree d. Li et al. first consider a general form for $c(x)$, as a product of distinct irreducible polynomials $c_j(x)$ over $\mathbb{F}_q$ and later in the construction of the scheme, they let the degree of every $c_j(x)$ be 1. Therefore, we also take

$$c(x) = \prod_{j=1}^{d} c_j(x) = \prod_{j=1}^{d} (x - \beta_j)$$

where $\beta_1, \ldots, \beta_d \in \mathbb{F}_q$ are distinct.

We consider the multiplicative group of units of the quotient ring $\mathbb{F}_q[x]/(c(x))$, whose structure is governed by the Chinese Remainder Theorem:

$$(\mathbb{F}_q[x]/(c(x)))^* \cong \prod_{j=1}^{d} (\mathbb{F}_q[x]/(c_j(x)))^* .$$

Since every component is a quotient by an irreducible polynomial, it's a field and we have $\mathbb{F}_q[x]/(c_j(x)) \cong \mathbb{F}_q$. Thus, the full ring $\mathbb{F}_q[x]/(c(x))$ is isomorphic to a direct product of d copies of $\mathbb{F}_q$, and likewise for the multiplicative group.

Now, let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ be distinct elements and disjoint from the set $\{\beta_1, \ldots, \beta_d\}$, which is why we required the condition $q \geq n + d$. Let $\mathbf{a} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_q^n$.

Define a group homomorphism $\varphi : \mathbb{Z}^n \to (\mathbb{F}_q[x]/(c(x)))^*$ by

$$\varphi(u_1, u_2, \ldots, u_n) = \prod_{i=1}^{n} (x - \alpha_i)^{u_i} \pmod{c(x)}.$$

Similarly, for each $1 \leq j \leq d$, define

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

Since $c_j(x) = x - \beta_j$, this simplifies to

$$\varphi_j(u_1, \ldots, u_n) = \prod_{i=1}^{n} (\beta_j - \alpha_i)^{u_i} \in \mathbb{F}_q^*.$$

Thus, the map φ can be viewed as a tuple of evaluations $(\varphi_1, \ldots, \varphi_d)$, and the image of $\mathbb{Z}^n$ under φ lies in $(\mathbb{F}_q^*)^d$, embedded via the Chinese Remainder Theorem.

We impose the following condition on the elements α_i and β_j:

Condition 1. *The set of polynomials $(x - \alpha_i)$ for $1 \leq i \leq n$ generates the multiplicative group of the ring $\mathbb{F}_q[x]/(c(x))$. In other words, for all $1 \leq j \leq d$ the set of values $\{(\beta_j - \alpha_i)\}$ generates the multiplicative group $\mathbb{F}_q^*$.*

Under Condition 1, the homomorphisms φ_j and φ are surjective. We define the lattice

$$\mathcal{L}_{\mathbf{a},c(x)} = \ker(\varphi) = \left\{ (u_1, u_2, \ldots, u_n) \in \mathbb{Z}^n : \prod_{i=1}^{n} (x - \alpha_i)^{u_i} \equiv 1 \pmod{c(x)} \right\}.$$

By definition, this is a sublattice of $\mathbb{Z}^n$. Since each $\varphi_j(x - \alpha_i) = \beta_j - \alpha_i$ is non-zero and lies in $\mathbb{F}_q^*$, which is cyclic of order $q - 1$, it follows that all image values have finite order dividing $q - 1$. Therefore, $(q-1)\mathbb{Z}^n \subseteq \mathcal{L}_{\mathbf{a},c(x)} \subseteq \mathbb{Z}^n$.

The ring $\mathbb{F}_q[x]/(c(x))$ is isomorphic to the direct product $\mathbb{F}_q^d$, and hence its multiplicative group is isomorphic to $(\mathbb{F}_q^*)^d$. Under Condition 1, the map φ is surjective, so by the First Isomorphism Theorem,

$$\mathbb{Z}^n / \mathcal{L}_{\mathbf{a},c(x)} \cong (\mathbb{F}_q^*)^d \cong (\mathbb{Z}/(q-1)\mathbb{Z})^d.$$

It follows that $\mathcal{L}_{\mathbf{a},c(x)}$ is a full-rank lattice in $\mathbb{Z}^n$ (i.e., of rank n) and has volume $(q-1)^d$.

The following lemma from the original paper (Lemma 3.2 (iii) [LLXY20] is used to establish the minimum distance of the lattice.

Lemma 3.1. *The lattice $\mathcal{L}_{\mathbf{a},c(x)}$ defined above satisfies $\lambda_1(\mathcal{L}_{\mathbf{a},c(x)}) \geq \sqrt{d}$. Moreover, $\lambda_1(\mathcal{L}_{\mathbf{a},c(x)}) = \sqrt{d}$ if and only if there exists a lattice point $\mathbf{v}$ of the form $\mathbf{v} = \pm\mathbf{v}_0$, where $\mathbf{v}_0$ has d non-zero entries which are all equal to 1.*

Proof. Let $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ be a non-zero vector in $\mathcal{L}_{\mathbf{a},c(x)}$. Define the sets $I = \{v_i > 0 : 1 \leq i \leq n\}$ and $J = \{v_j < 0 : 1 \leq j \leq n\}$. By the definition of $\mathcal{L}_{\mathbf{a},c(x)}$, we have

$$\prod_{i\in I}(x-\alpha_i)^{v_i}\prod_{j\in J}(x-\alpha_j)^{v_j} \equiv 1 \pmod{c(x)}.$$

Rearranging gives $\prod_{i\in I}(x-\alpha_i)^{v_i} - \prod_{j\in J}(x-\alpha_j)^{-v_j} \equiv 0 \pmod{c(x)}$. Hence, the non-zero polynomial $\prod_{i\in I}(x-\alpha_i)^{v_i} - \prod_{j\in J}(x-\alpha_j)^{-v_j}$ is divisible by $c(x)$. Therefore, its degree is at least d, so either $\sum_{i\in I} v_i \geq d$ or $\sum_{j\in J}(-v_j) \geq d$. Consequently, $\sum_{k=1}^{n} |v_k| \geq d$. It follows that

$$\|\mathbf{v}\|_2 = \sqrt{\sum_{k=1}^{n} v_k^2} \geq \sqrt{\sum_{k=1}^{n} |v_k|} \geq \sqrt{d}.$$

Thus, $\lambda_1(\mathcal{L}_{\mathbf{a},c(x)}) \geq \sqrt{d}$.

If there exists a lattice point with d non-zero entries which are either all 1 or -1, then it is clear that $\lambda_1(\mathcal{L}_{\mathbf{a},c(x)}) = \sqrt{d}$. Conversely, assume $\lambda_1(\mathcal{L}_{\mathbf{a},c(x)}) = \sqrt{d}$. Then there exists a non-zero lattice point $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ in $\mathcal{L}_{\mathbf{a},c(x)})$ such that $\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2} = \sqrt{d}$. Since we already know that $\deg(\prod_{i\in I}(x-\alpha_i)^{v_i} - \prod_{j\in J}(x-\alpha_j)^{-v_j}) \geq d$, we must have either $I = \emptyset$ and $\sum_{j\in J} -v_j = d$ or $J = \emptyset$ and $\sum_{i\in I} v_i = d$. This forces that either $v_i = 1$ for all $i \in I$ or $v_j = -1$ for all $j \in J$. □

Remark 3.1. *By Lemma 3.1, the lattice $\mathcal{L}_{\mathbf{a},c(x)}$ achieves minimum norm $\lambda_1(\mathcal{L}_{\mathbf{a},c(x)}) = \sqrt{d}$ if there exist distinct indices $i_1, \ldots, i_d \in [n]$ such that $\prod_{j=1}^{d}(x-\alpha_{i_j}) = 1 + c(x)$ or $\prod_{j=1}^{d}(x-\alpha_{i_j})^{-1} = 1 + c(x)$.*

Hence, there are at most $2\binom{n}{d}$ such degree d monic polynomials among the total q^d possibilities, giving

$$\Pr\left[\lambda_1(\mathcal{L}_{\mathbf{a},c(x)}) = \sqrt{d}\right] \leq \frac{2\binom{n}{d}}{q^d} < \frac{1}{d!}.$$

For the parameter ranges considered, this probability is negligible, so $\lambda_1(\mathcal{L}_{\mathbf{a},c(x)})$ is expected to exceed $\sqrt{d}$. In what follows, we therefore use Gaussian heuristic estimates for the minimum distance when an exact value is unavailable.

As we discussed in Section 3.1.1, a basis for $\mathcal{L}_{\mathbf{a},c(x)}$ can be computed in polynomial time in a straightforward way, by reducing to the computation of many discrete logarithms in $\mathbb{F}_q^*$.

Li et al. [LLXY20] make an assumption that is essentially equivalent to the Hermite normal form of a basis of the lattice having the form

$$\mathbf{B}_{\mathbf{a},c(x)} = \begin{pmatrix} \mathbf{I}_{n-d} & \mathbf{G}' \\ 0 & (q-1)\mathbf{I}_d \end{pmatrix}, \tag{3.2}$$

for some $(n-d) \times d$ matrix $\mathbf{G}'$. Here $\mathbf{I}_t$ denotes a $t \times t$ identity matrix. This property is implied by Condition 1. The paper [LLXY20] contains a very complicated algorithm to construct $\mathbf{G}'$, which involves constructing an invertible matrix and taking various discrete logarithms. See Proposition 3.3 of [LLXY20]. But it is equivalent to computing any basis of the lattice, by computing multidimensional discrete logarithms in $(\mathbb{F}_q^*)^d$, and then applying the Hermite normal form. The matrix $\mathbf{G}'$ is the public key for the LLXY encryption scheme and the secret key will become clear in Section 3.2.3.

3.2.2 The LLXY Lattice Decoding

To generate the trapdoor function, we first fix the public parameters q, n, d according to the desired security level. The trapdoor consists of the underlying group defined by the relation lattice, which includes the polynomial $c(x)$ and the vector $\mathbf{a} = (\alpha_1, \ldots, \alpha_n)$.

The inversion of the trapdoor function follows the decoding algorithm proposed by Ducas and Pierrot [DP19] for relation lattices.

Trapdoor Function

Let $\mathbf{G}'$ be as in Equation (3.2) and set $\mathbf{G} = \begin{pmatrix} \mathbf{I}_{n-d} & \mathbf{G}' \end{pmatrix}$. Let $\mathbf{m} \in (\mathbb{Z}/(q-1)\mathbb{Z})^{n-d}$ be a uniformly random message vector and let $\mathbf{e} \in \{0,1\}^n$ be an error vector satisfying $\|\mathbf{e}\|_1 < d$. The trapdoor function f is defined as

$$\mathbf{t} = f(\mathbf{m}, \mathbf{e}) = \mathbf{m}\mathbf{G} + \mathbf{e} \pmod{q-1}. \tag{3.3}$$

This construction mirrors the McEliece code-based encryption scheme, where the trapdoor function hides the structure of the code using small error vectors. In our setting, $\mathbf{G}$ is derived from the basis of the polynomial lattice $\mathcal{L}_{\mathbf{a},c(x)}$, and the function essentially encodes the message $\mathbf{m}$ with an error vector $\mathbf{e}$.

Without access to the trapdoor information, specifically the polynomial $c(x)$ and the vector $\mathbf{a} = (\alpha_1, \ldots, \alpha_n)$, inverting f amounts to recovering the error vector $\mathbf{e}$, or equivalently, finding a lattice vector $\mathbf{v} \in \mathcal{L}_{\mathbf{a},c(x)}$ such that $\|\mathbf{t} - \mathbf{v}\| = \sqrt{d-1}$. This is a

Bounded Distance Decoding (BDD) problem in the polynomial lattice $\mathcal{L}_{\mathbf{a},c(x)}$, which is generally believed to be computationally hard.

However, a party holding the trapdoor can efficiently recover $\mathbf{e}$ using the decoding algorithm proposed by Ducas and Pierrot [DP19], which exploits the structure of the relation lattice to solve the BDD instance efficiently.

Efficient Decoding

The central insight, inspired by the work of Ducas and Pierrot [DP19], is the existence of an efficient Bounded Distance Decoding (BDD) algorithm for certain types of error vectors. Specifically, consider a received vector $\mathbf{t} \in \mathbb{Z}^n$ of the form $\mathbf{t} = \mathbf{v} + \mathbf{e}$, where $\mathbf{v} = \mathbf{mG} \in \mathcal{L}_{\mathbf{a},c(x)}$ is a lattice vector and $\mathbf{e} \in \{0,1\}^n$ is a sparse error vector with ℓ_1-norm $\|\mathbf{e}\|_1 \leq d-1$. Since $\mathbf{e}$ is a binary vector, the ℓ_1-norm is equivalent to the Hamming weight.

Given $\mathbf{t}$, and the trapdoor information consisting of $\mathbf{a} = (\alpha_1, \ldots, \alpha_n)$ and the polynomial $c(x)$ of degree d, there exists an efficient algorithm to recover $\mathbf{e}$ and hence $\mathbf{m}$.

The decoding algorithm relies on the following algebraic observation. Let $\mathbf{t} = (t_1, \ldots, t_n)$, $\mathbf{v} = (v_1, \ldots, v_n)$ and $\mathbf{e} = (e_1, \ldots, e_n)$ and write each component as $t_i = v_i + e_i$. Then,

$$\prod_{i=1}^{n}(x-\alpha_i)^{t_i} = \prod_{i=1}^{n}(x-\alpha_i)^{v_i+e_i} = \prod_{i=1}^{n}(x-\alpha_i)^{v_i} \cdot \prod_{i=1}^{n}(x-\alpha_i)^{e_i}.$$

Since $\mathbf{v} \in \mathcal{L}_{\mathbf{a},c(x)}$, we have

$$\prod_{i=1}^{n}(x-\alpha_i)^{v_i} \equiv 1 \pmod{c(x)},$$

and therefore,

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

If the ℓ_1-norm $\|\mathbf{e}\|_1 = \sum_{i=1}^{n}|e_i| < d$, then the degree of the right-hand side polynomial is strictly less than $\deg(c(x)) = d$. In this case, its factorisation over the ring $\mathbb{F}_q[x]$ is the same as modular factorisation. Therefore, the set of linear factors $(x-\alpha_i)$ appearing in the factorisation reveals the support of $\mathbf{e}$, i.e., the positions where $e_i = 1$.

This decoding process is efficient due to the smoothness in $\mathbb{F}_q[x]$. The original LLXY paper deals with binary error vectors and uses polynomial factorisation. Hence, there is a unique factorisation (i.e., unique decoding) and therefore no decryption failure is possible for the algorithm.

Indeed, a simple trick from [CR84] allows decoding up to $\|\mathbf{e}\|_1 = d$ by noting that $\prod_{i=1}^{n}(x-\alpha_i)^{e_i}$ is monic (See Section IV in [CR84]). In particular, one can add a monic degree d polynomial $x^d - r(x) \equiv 0 \pmod{c(x)}$ to the polynomial $\prod_{i=1}^{n}(x-\alpha_i)^{t_i} \pmod{c(x)}$, and obtain a monic degree d polynomial that matches $\prod_{i=1}^{n}(x-\alpha_i)^{e_i}$, and hence can be uniquely factored to recover the error support.

Once $\mathbf{e}$ is determined, $\mathbf{t} - \mathbf{e} = \mathbf{mG} = (\mathbf{m}, -\mathbf{mG}')$. Therefore, the first $n-d$ entries give the $\mathbf{m} \pmod{q-1}$.

3.2.3 The LLXY Encryption Scheme

Upon building a trapdoor function that admits efficient decoding, the authors of [LLXY20] construct a public key encryption scheme, known as the LLXY encryption scheme.

The *public key* of the scheme consists of the $(n-d) \times d$ matrix $\mathbf{G}'$, which appears in the generator matrix

$$\mathbf{G} = \begin{pmatrix} \mathbf{I}_{n-d} & \mathbf{G}' \end{pmatrix},$$

along with the system parameters n, d, and the prime q.

The *private key* (or trapdoor) consists of the polynomial $c(x) \in \mathbb{F}_q[x]$ and the vector of distinct elements $\mathbf{a} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_q^n$, which define the relation lattice $\mathcal{L}_{\mathbf{a},c(x)}$ and enable efficient decoding.

To encrypt, the sender selects a random message vector $\mathbf{m} \in (\mathbb{Z}/(q-1)\mathbb{Z})^{n-d}$ and the error vector $\mathbf{e} \in \{0,1\}^n$ with ℓ_1-norm $\|\mathbf{e}\|_1 < d$, and computes the ciphertext

$$\mathsf{c} = f(\mathbf{m}, \mathbf{e}) = \mathbf{mG} + \mathbf{e} \pmod{q-1}.$$

Given the trapdoor information, the receiver can efficiently recover $\mathbf{e}$ and consequently $\mathbf{m}$ using the decoding algorithm discussed in the previous section.

The ciphertext consists of the vector $\mathbf{t} \in (\mathbb{Z}/(q-1)\mathbb{Z})^n$, and hence the ciphertext size is $n\log_2(q-1)$ bits. For a 128-bit security level, [LLXY20] suggests the parameter choices $n = 500, d = 43$, and $q = 29599$, which yield ciphertexts of size $500 \times \log_2(29598) = 7500$ bits (938 bytes)

The authors note that this construction "can be viewed as a lattice analogue of the Goppa-code-based McEliece cryptosystem." However, we prefer to think of it as an implementation of the McEliece cryptosystem using a code derived from lattices, in particular, from the relation lattice $\mathcal{L}_{\mathbf{a},c(x)}$.

Section VI of [LLXY20] discusses CCA (chosen ciphertext attack) security, proposing a padding scheme and decryption checks to achieve robustness. In Section VII, the authors further claim that the resulting scheme achieves CCA security under their proposed design.

We now explain the padding scheme and the decryption checks propose to achieve the CCA security.

Let $s = \lceil \log_2(q-1) \rceil$ denote the number of bits needed to represent any element of the ring $\mathbb{Z}/(q-1)\mathbb{Z}$. This parameter determines the bit-length of each component in the binary decomposition of the encoded message. Let the plaintext message be a bit string $P \in \{0,1\}^{n-d}$.

Encryption: The $n-d$ bit plaintext message P is encoded into the input $\mathbf{m}$ for the trapdoor function f as follows. First, a random vector $\mathbf{z} \in \{0,1\}^{n-d}$ is sampled. This serves as a one-time pad that will be used to mask the message via a bitwise XOR operation. In addition, an error vector $\mathbf{e} \in \{0,1\}^n$ is sampled uniformly at random such that its ℓ_1-norm satisfies $\|\mathbf{e}\|_1 = d-1$ (this guarantees $\|\mathbf{e}\|_1 < d$ deterministically).

Next, define a set of auxiliary vectors that together encode the message and relevant randomness. The first component is defined as $\mathbf{m}^{(0)} = P \oplus \mathbf{z}$, which masks the plaintext using the sampled randomness. The second component is simply $\mathbf{m}^{(1)} = \mathbf{z}$, and define the third component as $\mathbf{m}^{(2)} = \mathsf{H}(P\|\mathbf{z}\|\mathbf{e})$, where H is a cryptographic hash function mapping its input to a bit string of length $n-d$. To complete the encoding, for each $j = 3, \ldots, s-1$, generate independent random bit strings $\mathbf{m}^{(j)} \in \{0,1\}^{n-d}$. These vectors serve as padding to reach the full bit length required for representation in $\mathbb{Z}/(q-1)\mathbb{Z}$.

Now assemble the full encoded message vector $\mathbf{m} \in \mathbb{Z}^{n-d}$ using these bit strings. Specifically, for each index $1 \le i \le n-d$, we define

$$\mathbf{m}_i = \sum_{j=0}^{s-1} 2^j \mathbf{m}_i^{(j)},$$

so that the vector $\mathbf{m}^{(j)}$ is the vector of j-th bits of $\mathbf{m}$.

Finally, the ciphertext will be the output of $f(\mathbf{m}, \mathbf{e})$ in $\mathbb{Z}_{q-1}^n$ for the public generator matrix $\mathbf{G}$ as follows

$$\mathsf{c} = \mathbf{m}\mathbf{G} + \mathbf{e} \pmod{(q-1)}.$$

Decryption: This decryption relies on the knowledge of the trapdoor information associated with the public matrix $\mathbf{G}$, which allows efficient decoding of the ciphertext despite the presence of a small error vector.

Given a ciphertext $\mathsf{c} \in (\mathbb{Z}/(q-1)\mathbb{Z})^n$, the receiver uses the trapdoor information and the decoding algorithm to recover $(\mathbf{m}, \mathbf{e})$ such that $\mathbf{m} = f^{-1}(\mathsf{c})$.

Once the vector $\mathbf{m}$ has been recovered, the receiver performs a binary decomposition of each of its components to extract the embedded bit strings. Specifically, for each $1 \le i \le$

$n-d$, the s-bit binary representation of $\mathbf{m}_i$ is obtained, yielding the vectors $\mathbf{m}^{(j)} \in \{0,1\}^{n-d}$ for $j = 0, 1, \ldots, s-1$. These bit strings correspond to the same components used during the encoding step, where $\mathbf{m}^{(0)}$ masked the plaintext, $\mathbf{m}^{(1)}$ contained the random vector, and $\mathbf{m}^{(2)}$ held the hash-based validity check.

With the vectors $\mathbf{m}^{(0)}$ and $\mathbf{m}^{(1)}$ recovered, the plaintext can be reconstructed as,

$$P = \mathbf{m}^{(0)} \oplus \mathbf{m}^{(1)},$$

since the original message P was masked using a bitwise XOR with the random string $\mathbf{z} = \mathbf{m}^{(1)}$ during encryption.

To verify the validity of the ciphertext, the receiver computes the hash value using the reconstructed plaintext P, the recovered randomness $\mathbf{m}^{(1)}$, and the error vector $\mathbf{e}$. That is, check whether

$$\mathbf{m}^{(2)} = \mathsf{H}(P \| \mathbf{m}^{(1)} \| \mathbf{e})$$

is satisfied. If the computed hash matches the recovered $\mathbf{m}^{(2)}$, the decryption is successful and the plaintext P is accepted. If the check fails, the ciphertext is rejected.

3.2.4 Extending the Error Distribution

In the original LLXY setting, the error vectors were binary, i.e., $\mathbf{e} \in \{0,1\}^n$, which enabled a relatively simple decoding procedure based on polynomial factorisation. However, Lapiha [Lap21] proposed an extension of the error distribution from binary values to ternary values, specifically $\mathbf{e} \in \{-1,0,1\}^n$. In the ternary case, Lapiha assumed that the error contains the same number of positive and negative coordinates. The factorisation technique used for decoding in the binary case no longer suffices in the presence of negative error components.

To address this challenge, the decoding algorithm adopts a strategy inspired by the approach used by Ducas and Pierrot for decoding discrete error vectors over the integers $\mathbb{Z}^n$ (see Section 3.2.2 of [DP19]). Following this framework, they use continued fractions for polynomials.

Concretely, the goal is to reconstruct the expression

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

as a quotient, where $e_i \in \{-1,0,1\}$ represents the error at position i, and $c(x)$ is a known modulus polynomial. Since the exponents e_i may now be positive or negative, the right-hand

side corresponds to a rational function $\frac{U(x)}{V(x)}$ where the numerator $U(x)$ and the denominator $V(x)$ capture the positive and negative exponents, respectively.

To recover such a rational function representation from a given polynomial modulo $c(x)$, one can use a continued fraction expansion based on the extended Euclidean algorithm. Lapiha refers to the analysis by Khodadad and Monagan [KM06], which describes an efficient method to recover rational approximations of the form

$$\frac{U(x)}{V(x)} \equiv g(x) \pmod{c(x)},$$

under the constraint that the sum of the degrees satisfies $\deg(U) + \deg(V) < \frac{\deg(c(x))}{2}$. This restriction ensures the uniqueness of the recovered fraction. To guarantee successful decoding, Lapiha proposes to restrict the error vectors $\mathbf{e}$ to those satisfying a bounded ℓ_1-norm condition:

$$\|\mathbf{e}\|_1 = \sum_{i=1}^{n} |e_i| \leq \left\lfloor \frac{d}{2} \right\rfloor,$$

where d denotes the degree of the polynomial $c(x)$. This bound ensures that the degrees of the numerator and denominator in the rational reconstruction remain sufficiently small.

However, we observe that while this restriction efficiently avoids decoding failures, it may be overly conservative in practice. In particular, enforcing a strict $\|\mathbf{e}\|_1 \leq \lfloor d/2 \rfloor$ is especially restrictive in the LLXY parameter choices, where $n \gg d$. In this setting, the condition may significantly reduce the number of acceptable error vectors, thus weakening the overall performance and security level of the scheme. We argue that it is possible to relax this constraint without compromising correctness in most cases.

In our variant of the scheme, we therefore consider a looser bound on the error weight, namely

$$\|\mathbf{e}\|_1 < d.$$

This allows for a larger error space and improves the flexibility of parameter selection, while still maintaining unique decoding under typical parameter regimes. We provide a more detailed discussion and formal analysis of this relaxed decoding condition in the next chapter.

3.3 Security Analysis of the Original LLXY Scheme

Li, Ling, Xing, and Yeo discussed several potential attacks against the proposed cryptographic scheme in their original work [LLXY20]. This section provides a summary

of the main attacks presented in the LLXY paper, as well as a set of decoding attacks later proposed by Lapiha [Lap21], which further assess the robustness of the scheme. In addition, we introduce a new adaptive chosen-ciphertext (CCA) attack that targets the CCA protection mechanism of the LLXY scheme. This novel attack, detailed in Section 3.3.3, highlights previously overlooked vulnerabilities and motivates a re-evaluation of the scheme's CCA security.

Li et al. discuss several potential attacks targeting the trapdoor function underlying their scheme. These include structural attacks that rely on exhaustive search over the error vectors and trapdoor information, aiming to recover the secret keys.

In Section 5.4 of [LLXY20], the authors discuss that one particular concern arises when the polynomial $c(x)$ used in the construction is irreducible over $\mathbb{F}_q$. In such cases, certain algebraic structures may be exploited, potentially weakening the scheme. To mitigate this risk and to avoid attacks inspired by the Chor-Rivest cryptosystem [CR84], the scheme construction deliberately chooses $c(x)$ as a product of linear polynomials.

Among the various attacks considered, the primary one of interest is the embedding attack, which attempts to solve the BDD problem in the associated lattice. This attack translates the problem of decoding a ciphertext into a BDD instance, and uses lattice reduction techniques to find the error vector. We provide a detailed discussion of the lattice attack in the following subsection.

3.3.1 Lattice Attacks

Li, Ling, Xing, and Yeo [LLXY20] analysed a lattice-based attack against the LLXY scheme using Kannan's embedding technique [Kan87]. In this setting, the public key matrix $\mathbf{G}'$ can be used to construct a lattice basis of the following form:

$$\mathbf{B} = \begin{pmatrix} \mathbf{I}_{n-d} & \mathbf{G}' \\ 0 & (q-1)\mathbf{I}_d \end{pmatrix},$$

where $\mathbf{I}_{n-d}$ and $\mathbf{I}_d$ are identity matrices of appropriate dimensions. A ciphertext can be viewed as a lattice vector with a small additive error. Hence, recovering the plaintext amounts to solving the Bounded Distance Decoding (BDD) problem with respect to the ℓ_1-norm.

A standard approach to solving BDD problems is via the embedding technique, which transforms the BDD instance into a unique Shortest Vector Problem (uSVP) (see Definition 2.10), allowing the use of lattice reduction algorithms such as BKZ.

To perform this transformation, Lyubashevsky and Micciancio [LM09] first showed that solving a $\text{BDD}_{1/(2\gamma)}$ problem (subscript corresponds to the decoding radius) instance reduces to solving a uSVP_γ instance for any $\gamma \geq 1$. This reduction was later improved by Bai et al. [BSW16] and demonstrated that for any polynomially bounded $\gamma \geq 1$, BDD with decoding radius approximately $1/(\sqrt{2}\gamma)$ reduces to uSVP with approximation factor γ.

To solve the resulting uSVP instance, Li et al. adopt the *primal lattice attack* strategy outlined by Alkim et al. (see Section 6.3 of [ADPS16]), and further refined in the work of Albrecht et al. [AGVW17]. Their analysis models the performance of BKZ lattice reduction using the Geometric Series Assumption (GSA)(Definition 2.16). Under this assumption, the lengths of the Gram-Schmidt orthogonalised basis vectors $\mathbf{b}_i^*$ are given by $\|\mathbf{b}_i^*\|_2 = \delta^{n-2i}(\det \mathcal{L})^{1/n}$, where δ is the root-Hermite factor, approximated as

$$\delta = \left(\frac{(\pi\beta)^{1/\beta} \cdot \beta}{2\pi e}\right)^{\frac{1}{2(\beta-1)}},$$

and β denotes the BKZ block size (see Section 3.2 of [APS15]).

The key idea is that the unique short error vector $\mathbf{e}$ will be detected by BKZ if the projection of $\mathbf{e}$ onto the vector space spanned by the last β Gram-Schmidt vectors is shorter than $\mathbf{b}_{n-\beta}^*$. The projected norm of the error vector $\mathbf{e}$ in the embedding is expected to be

$$\|(\mathbf{e}, 1)\|_2 \sqrt{\frac{\beta}{n}}.$$

Therefore, the embedding attack is successful if and only if the following inequality is satisfied

$$\sqrt{\frac{\beta}{n}}\|(\mathbf{e}, 1)\|_2 \leq \delta^{2\beta-n}(\det \mathcal{L})^{1/n}. \tag{3.4}$$

To ensure that the LLXY encryption scheme resists such embedding attacks, Li et al. select parameters so that any block size β large enough to satisfy condition (3.4) would require a BKZ reduction whose computational cost exceeds 2^λ, where λ denotes the desired security level in bits. Consequently, although the embedding condition may hold, the corresponding BKZ cost becomes computationally infeasible at the intended security level.

However, the authors do not consider other lattice attack strategies, such as dual attacks or hybrid methods, which have since been shown to be more effective in related contexts. In Section 5.1, we provide a detailed comparison and demonstrate that these alternative attacks outperform the embedding/unique-SVP approach in practice.

3.3.2 Decoding Attacks

Lapiha [Lap21] was the first to demonstrate the effectiveness of decoding attacks against the LLXY cryptosystem. They propose cryptanalysis of the LLXY scheme using Information Set Decoding (ISD) techniques.

The core of the decoding attack relies on the parity check matrix $\mathbf{H}$ associated with the public generator matrix $\mathbf{G}$, which is typically given in systematic form as $\mathbf{G} = \begin{pmatrix} \mathbf{I}_{n-d} & \mathbf{G}' \end{pmatrix}$. Given the generator matrix $\mathbf{G}$, one can compute the parity check matrix $\mathbf{H} = \begin{pmatrix} -\mathbf{G}' \\ \mathbf{I}_d \end{pmatrix}$ such that $\mathbf{GH} = \mathbf{0}$ (see Section 2.5). Multiplying the ciphertext c by the parity check matrix $\mathbf{H}$ yields

$$\mathsf{c}\mathbf{H} = (\mathbf{mG} + \mathbf{e})\mathbf{H} = \mathbf{eH}.$$

Thus, the decoding problem reduces to the classic syndrome decoding problem (Definition 2.23), where the syndrome is given by $\mathbf{s} = \mathsf{c}\mathbf{H} = \mathbf{eH}$. Using the properties of the LLXY scheme, it is known that a binary error vector $\mathbf{e}$ satisfies the bound $\|\mathbf{e}\|_1 \leq d - 1$.

Lapiha discusses three decoding attack strategies: a basic Meet-in-the-Middle (MitM) attack, an Information Set Decoding (ISD) attack, and a hybrid attack that combines the two approaches.

In the ISD attack, the parity check matrix $\mathbf{H}$ is first transformed into systematic form as shown above, and the error vector $\mathbf{e}$ is partitioned as $\mathbf{e} = \begin{pmatrix} \mathbf{e}' & \mathbf{e}'' \end{pmatrix}$. The syndrome equation then becomes

$$\mathbf{s} = -\mathbf{e}'\mathbf{G}' + \mathbf{e}''.$$

The attack proceeds by enumerating all possible low-weight candidates for $\mathbf{e}'$, computing the corresponding $\mathbf{e}'' = \mathbf{s} + \mathbf{e}'\mathbf{G}'$, and checking whether the combined weight of $(\mathbf{e}', \mathbf{e}'')$ matches the known error weight.

Their meet-in-the-middle attack operates without relying on the systematic form of $\mathbf{H}$. Instead, the matrix $\mathbf{H}$ and the error vector $\mathbf{e}$ are partitioned into two halves, under the assumption that the error weight is evenly distributed between them.

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}' \\ \mathbf{H}'' \end{pmatrix}, \quad \mathbf{e} = \begin{pmatrix} \mathbf{e}' & \mathbf{e}'' \end{pmatrix}.$$

The idea is if an attacker finds two error vectors $\mathbf{e}'$ and $\mathbf{e}''$ such that $\mathbf{e}'\mathbf{H}'$ and $\mathbf{s} - \mathbf{e}''\mathbf{H}''$ collide, then the concatenation of $\mathbf{e}'$ and $\mathbf{e}''$ gives an error vector $\mathbf{e}$ satisfying the equation $\mathbf{s} = \mathbf{e}\mathbf{H}$.

The final, combined attack merges the MitM and ISD approaches and considers approximate collisions related to the nearest neighbour search problem introduced by May and Ozerov [MO15]. We will explore this in greater detail in Chapter 5.

It is worth noting that Lapiha's cryptanalysis is based on an earlier variant of LLXY [LLXY17] that solves the Closest Vector Problem (CVP) rather than the Bounded Distance Decoding problem. Their analysis revealed that the parameter sizes suggested in [LLXY17] fall short of providing the desired post-quantum security level. In fact, their attacks require an average computational effort of approximately 2^{25} and 2^{26} operations for the smallest and largest parameter sets, respectively, rendering those parameter choices insecure.

3.3.3 Adaptive Attack

In Section 3.2.3, we described the LLXY scheme and its claimed CCA protections. In this section, we show that these protections are insufficient to prevent an adaptive attack. The root of the issue lies in the fact that the decryption procedure only checks the least significant bits (LSBs) of the message vector $\mathbf{m}$, while ignoring the higher-order bits.

Suppose an attacker receives a valid ciphertext $\mathsf{c} = \mathbf{m}\mathbf{G} + \mathbf{e} \pmod{q-1}$. Let $m_i^{(j)}$ denote the j-th bit of the i-th coordinate $\mathbf{m}_i$ of $\mathbf{m}$, so that $\mathbf{m}_i = \sum_{j=0}^{s-1} 2^j m_i^{(j)}$. In a correctly padded message, the bits of $\mathbf{m}$ satisfy the following structure:

$$\mathbf{m}^{(0)} = P \oplus z, \quad \mathbf{m}^{(1)} = z, \quad \mathbf{m}^{(2)} = \mathsf{H}(P \| z \| \mathbf{e}),$$

where P is the plaintext, z is a random vector, and H is a hash function.

Note that $\mathbf{m} \in (\mathbb{Z}/(q-1)\mathbb{Z})^{n-d}$, so each $\mathbf{m}_i$ is determined modulo $q-1$ and is treated as an integer in the range $\{0, 1, \ldots, q-2\}$.

Now consider a random index i. If any of the higher bits $m_i^{(j)} = 0$ for some $3 \leq j < s-1$, then we know that some high-value binary weight is not yet used in $\mathbf{m}_i$. This means that $\mathbf{m}_i$ is smaller than the maximum possible value under modulo $(q-1)$. For example, if $m_i^{(3)} = 0$, then there's room to add 8 without exceeding $q-1$. i.e., $\mathbf{m}_i + 8 < q - 1$ holds. In this case, we have $\mathbf{m}_i < q - 9$, meaning that $\mathbf{m}_i$ can takes values in the range $\{0, \ldots, q-10\}$.

This allows an attacker to form a modified ciphertext

$$\mathsf{c}' = \mathsf{c} + (0, 0, \ldots, 0, 8, 0, \ldots, 0)\mathbf{G},$$

where the value 8 is added at the i-th position. This operation corresponds to changing higher-order bits $m_i^{(j)}$ for $j \geq 3$, without affecting the lower-order bits that are actually checked during decryption. It means c' decrypts correctly to give the same plaintext P as c.

This observation enables a straightforward CCA attack: the attacker chooses a random index i, constructs such a modified ciphertext $\mathsf{c}' \neq \mathsf{c}$, and submits it to a decryption oracle. The returned message $\mathbf{m}'$ decrypts to the same original plaintext P.

The success probability of this attack for a randomly chosen index i is at least

$$\Pr(\mathbf{m}_i < q - 9) = \frac{q-9}{q-1} = 1 - \frac{8}{q-1},$$

which is at least $1/2$ for all $q \geq 17$. Hence, the probability that the attack succeeds for at least one index $i \in \{1, \ldots, n-d\}$ is overwhelming for any parameter set of practical interest.

A naive fix would be to include checks on all bits of $\mathbf{m}$ during decryption. However, it is generally preferable to adopt a well-established CCA-secure construction (e.g., Fujisaki–Okamoto (FO) transform) rather than attempt ad hoc protections. One should also note that the FO transform fundamentally changes the primitive, converting the underlying public-key encryption scheme into a key encapsulation mechanism (KEM). In light of this, we redesign the construction directly as a KEM and establish its CCA security within that framework in Section 4.4.

3.4 Future Directions

In this section, we present a new perspective on viewing the underlying lattice, which provides deeper insight into the structure and properties of the relation lattice. This structural understanding opens the door to new forms of cryptanalysis. Based on this viewpoint, we explore potential structural attack strategies that could be valuable in future cryptanalytic efforts.

3.4.1 New Formulation to Analyse the Lattice

We now present a reformulation of the relation lattice that eliminates the need for polynomial congruences and expresses the condition entirely within the finite field $\mathbb{F}_q$. This may facilitate further structural attacks and cryptanalysis.

Lemma 3.2. *Suppose* $c(x) = \prod_{j=1}^{d}(x - \beta_j)$ *splits completely over* $\mathbb{F}_q$ *into distinct linear factors. Then a vector* $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{Z}^n$ *is in the LLXY relation lattice if and only if*

$$\prod_{i=1}^{n}(\beta_j - \alpha_i)^{u_i} = 1 \quad \textit{for all } 1 \leq j \leq d. \tag{3.5}$$

We emphasise that the equality in (3.5) is in the finite field $\mathbb{F}_q$; thus, the formulation no longer involves polynomial congruences.

Proof. The relation lattice is defined as the set

$$\left\{(u_1, u_2, \ldots, u_n) \in \mathbb{Z}^n : \prod_{i=1}^{n}(x - \alpha_i)^{u_i} \equiv 1 \pmod{c(x)}\right\}.$$

By the definition of $c(x)$ and the Chinese Remainder theorem, congruence modulo $c(x)$ is equivalent to congruence modulo each linear factor $(x - \beta_j)$, for all $1 \leq j \leq d$.

For any polynomial $f(x)$, we have $f(x) \equiv f(\beta_j) \pmod{(x - \beta_j)}$. Applying this, we get

$$\prod_{i=1}^{n}(x - \alpha_i)^{u_i} \pmod{(x - \beta_j)} \equiv \prod_{i=1}^{n}(\beta_j - \alpha_i)^{u_i}.$$

Therefore, the original congruence holds modulo $c(x)$ if and only if

$$\prod_{i=1}^{n}(\beta_j - \alpha_i)^{u_i} = 1 \in \mathbb{F}_q \quad \text{for all } j = 1, \ldots, d.$$

This completes the proof. □

A similar formula also applies when $c(x)$ has higher degree irreducible factors, but we need to work with field extensions.

From Equation (3.5) we immediately notice that if $(\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_d)$ is a secret key, then so is $(\alpha_1 + \theta, \ldots, \alpha_n + \theta, \beta_1 + \theta, \ldots, \beta_d + \theta)$ for any $\theta \in \mathbb{F}_q$. This fact is already mentioned by LLXY (Remark 3.4 [LLXY20]). Hence, without loss of generality we may take $\beta_1 = 0$, giving us $n + d - 1$ unknowns.

One can also understand some properties of the lattice this way. Suppose the lattice contains a vector

$$(0, 0, \ldots, 0, w, 0, \ldots, 0)$$

where the non-zero value w is in the i-th position. Then $(\beta_j - \alpha_i)^w = 1$ for all $1 \leq j \leq d$. This implies that all the differences $(\beta_j - \alpha_i)$ lie in a multiplicative subgroup of $\mathbb{F}_q^*$ of order

dividing w. If $(q-1) \nmid w$ then, it follows that all the d elements $\beta_j - \alpha_i$ lie in a proper subgroup of $\mathbb{F}_q^*$. This is unlikely for "random" β_j and large d. Hence, in general (and this is observed experimentally) the lattice contains elements $(0, 0, \ldots, 0, w, 0, \ldots, 0)$ if and only if w is a multiple of $(q-1)$. Our Condition 1 explicitly ensures this is the case.

One can also consider sub-lattices of rank d. Let $I \subseteq \{1, \ldots, n\}$ be of size d and let $\mathcal{L}'$ be the set of vectors $\mathbf{v} \in \mathcal{L}$ such that $v_i \neq 0$ only if $i \in I$. In other words, $\mathcal{L}' = \mathcal{L} \cap \mathbb{Z}^d$ where $\mathbb{Z}^d$ is embedded into $\mathbb{Z}^n$ in the positions indexed by I. A vector $\mathbf{v}$ in $\mathcal{L}'$ corresponds to d equalities

$$\prod_{i \in I} (\beta_j - \alpha_i)^{v_i} = 1$$

for $1 \leq j \leq d$. Equivalently, we have a product of d elements that is equal to the identity in $(\mathbb{F}_q^*)^d$. Since the group $(\mathbb{F}_q^*)^d$ cannot be generated by fewer than d generators, one generally does not expect such relations among random elements. Hence, we typically expect that $\mathcal{L}' = (q-1)\mathbb{Z}^d$, and this is observed in practice when d and q are sufficiently large.

3.4.2 Structural attacks

A structural attack is a direct way to determine the secret key from the public key, bypassing generic lattice and code techniques. The original LLXY paper [LLXY20] already mentions a number of ideas for structural attacks. Our work started with trying to find new structural attacks. The results in Section 3.4.1 are a direct result of attempts at a structural attack, and we now build on some of the observations in that section.

Solving non-linear equations: Each vector in the lattice, via Equation (3.5) gives d non-linear equations (one for each value of j) in the $n+d-1$ unknowns. Consider a basis for the lattice in Hermite normal form as in Equation (3.2). The final d rows give no information about the private key, as they are simply the fact that $(\beta_j - \alpha_i)^{q-1} = 1$ for $n-d+1 \leq i \leq n$. The $(n-d)$-th row is more interesting as it is of the form $(0, ..., 0, 1, g'_1, \ldots, g'_d)$ where g'_i are entries from the last row of $\mathbf{G}'$ and corresponds to

$$(\beta_j - \alpha_{n-d}) = \prod_{i=1}^{d} (\beta_j - \alpha_{n-d+i})^{-g'_i}$$

for each $1 \leq j \leq d$. Hence, $n-d$ of the vectors in the basis are not trivial, and we have a system of $(n-d)d$ non-linear equations in the $n+d-1$ variables. The problem is that the equations are very highly non-linear, and we are not able to solve them directly (e.g., via Gröbner basis methods).

When a part of the secret key is known: The diagonal shape of the Hermite normal form has a further consequence. If just some elements are known, for example the $2d$ values $\beta_1, \ldots, \beta_d, \alpha_{n-d+1}, \ldots, \alpha_n$ then one can compute the remaining values $\alpha_{n-d}, \alpha_{n-d-1}, \ldots, \alpha_1$. This suggests an attack strategy to compute part of the secret key and then solve for the rest using these equations. The problem with such an attack is how to compute some initial values β_j and α_i.

It is instructive to consider a simpler problem: Suppose $c(x)$ is known to the attacker (equivalently, all the β_j known). Can the attacker determine the α_i? One can mount a "meet-in-the-middle" version of brute-force search over some subset of the α_i. Precisely, suppose we have a lattice vector u supported on the last $2k$ coordinates. In other words, $u_1 = u_2 = \cdots = u_{n-2k} = 0$. Then,

$$\prod_{i=1}^{k} (x - \alpha_{n-2k+i})^{u_{n-2k+i}} \equiv \prod_{i=1}^{k} (x - \alpha_{n-k+i})^{-u_{n-k+i}} \pmod{c(x)}$$

and one can compute a list of all q^k choices for the left-hand-side (trying all α_{n-2k+i}) and then try all q^k choices for the right-hand-side and see if there is a match. The problem is that we need to consider at least $k \geq d/2$ of the α_i, and so the attack needs at least $q^{d/2}$ steps to try all these α_i's. For our parameters that is already infeasible.

Note that one can guess a root β of $c(x)$ with reasonable probability (typically around $1/7$ with the parameters we are suggesting in Section 5.4). Hence, one can guess a low-degree factor of $c(x)$. But since no feasible attack is known even when the whole of $c(x)$ is given to the attacker, we do not investigate this further.

Finding the secret key: Another class of attacks that we studied is to write the problem as a system of polynomial equations and attempt to solve it to learn the secret key. The basic idea is that a short vector $(u_1, \ldots, u_n)$ in the lattice gives a polynomial equation

$$\prod_{u_i > 0} (x - \alpha_i)^{u_i} \equiv \prod_{u_i < 0} (x - \alpha_i)^{-u_i} \pmod{c(x)}. \tag{3.6}$$

The coefficients of the polynomial $c(x)$ and the values α_i are treated as unknown variables. For our parameters, it is easy to check that these systems are far beyond what can be feasibly solved (the number of variables is well over 500 and the degrees of the equations are at least the ℓ_1-norm of the shortest non-zero vector, which is at least d).

In summary, we claim that the above structural attacks are not feasible for the parameters we are proposing and none of these attacks are competitive with the direct lattice or ISD

attacks which we will discuss in the next chapter (Chapter 5). Nevertheless, we encourage others to work on this problem and try to find new structural attacks.

3.5 Summary

In this chapter, we explored an interesting family of lattices arising from the discrete logarithm problem, focusing in particular on a variant defined using polynomials. We examined an encryption scheme that uses a trapdoor function based on these polynomial lattices, and we demonstrated that the scheme is not CCA-secure using an adaptive attack, contrary to the original claims made by the authors.

We also reviewed the cryptanalysis techniques presented in the original paper, as well as related decoding attacks. Furthermore, we introduced a new formulation of the relation lattice, which offers additional structural insight and may enable future cryptanalytic approaches.

We believe the LLXY scheme warrants further scrutiny due to its potential advantages over both lattice-based and code-based cryptography. In the next chapter, we redefine the LLXY scheme as a key encapsulation mechanism and outline how to obtain CCA security using standard techniques.

Chapter 4

LLXY KEM

Contents

The original LLXY scheme is based on the McEliece cryptosystem [McE78], in which encryption relies on the use of a generator matrix. In this chapter, we instead use Niederreiter's cryptosystem [Nie86], which uses the parity-check matrix and thereby enables more compact ciphertexts while preserving security. We redefine LLXY as a Key Encapsulation Mechanism (KEM) and outline how to achieve Chosen Ciphertext Attack (CCA) security using standard transformation techniques.

4.1 Background & Motivation

In the previous chapter, we examined a McEliece encryption scheme based on $(q-1)$-ary codes derived from polynomial lattices. This naturally led us to investigate the dual formulation of the scheme, namely the Niederreiter cryptosystem, which offers certain advantages in terms of efficiency and ciphertext size.

In 1986, Niederreiter [Nie86] introduced a variant of the McEliece cryptosystem that employs a parity-check matrix for encryption, rather than the generator matrix used in the original McEliece proposal [McE78]. This structural change not only reduces ciphertext size but also simplifies the error decoding process. Niederreiter originally suggested the use of Reed-Solomon codes in his construction; however, these were later proven to be insecure in this context due to the work of Sidelnikov and Shestakov [SS92]. Despite this, the Niederreiter cryptosystem remains secure when instantiated with binary Goppa codes, and has been shown to be equivalent to the McEliece cryptosystem in terms of security when the same parameters are used [LDW94].

Originally, Niederreiter's system was designed to encode messages into the error vectors $\mathbf{e}$. However, modern cryptographic design principles favour using such encryption primitives as *Key Encapsulation Mechanisms* (KEMs) [Den03]. A KEM is a cryptographic primitive that allows anyone in possession of some party's public key to securely transmit a key to that party. A KEM can be viewed as a key-exchange protocol in which only a single message is transmitted.

Key Encapsulation. In many practical scenarios, full-fledged public-key encryption is unnecessary; instead, it suffices for two parties to establish a shared random session key. This functionality is provided by a Key Encapsulation Mechanism (KEM), as formalised by Cramer and Shoup [CS03].

In a KEM protocol, the sender who has access to the receiver's public key runs an encapsulation algorithm that outputs both a random session key (also known as the encapsulated key) and an associated ciphertext. The ciphertext is then sent to the receiver over potentially an untrusted channel. Upon receiving it, the receiver applies the decapsulation algorithm using their private key to recover the same session key. As a result, both parties obtain a shared random key suitable for use in subsequent cryptographic operations.

A robust security requirement for KEMs is chosen-ciphertext security (CCA security), which ensures resilience even against active adversaries. Specifically, it guarantees that an attacker, despite having access to a decapsulation oracle and being able to query it on ciphertexts of their choosing, cannot learn anything about the session key corresponding to a challenge ciphertext they did not submit to the oracle.

Once the key agreement is complete, the shared encapsulated key can be used as input to symmetric-key primitives, such as encryption schemes.

Motivated by these observations, we redefine the LLXY encryption scheme as a KEM, taking advantage of the structural benefits of the Niederreiter cryptosystem. In particular,

this approach enables the use of the parity-check matrix rather than the generator matrix, leading to more compact ciphertexts without compromising the underlying security of the scheme. This transformation not only aligns LLXY with modern cryptographic practices but also improves its practicality and efficiency in real-world deployments.

4.2 The Niederreiter Variant

In this section, we describe how to convert the LLXY encryption scheme into a Key Encapsulation Mechanism (KEM) using the parity-check matrix $\mathbf{H}$. Recall that in the LLXY scheme, the generator matrix is defined as

$$\mathbf{G} = \begin{pmatrix} \mathbf{I}_{n-d} & \mathbf{G}' \end{pmatrix} \in \mathbb{Z}_{q-1}^{(n-d)\times n},$$

and encryption involves a random error vector $\mathbf{e} \in \{0,1\}^n$ satisfying $\|\mathbf{e}\|_1 \leq d-1$, together with a message $\mathbf{m} \in \mathbb{Z}_{q-1}^{n-d}$. The ciphertext is computed as $\mathsf{c} = \mathbf{mG} + \mathbf{e} \pmod{q-1}$.

In the Niederreiter variant, we instead use the parity-check matrix $\mathbf{H}$ in standard form which satisfies $\mathbf{GH} = \mathbf{0} \pmod{q-1}$. As noted in Section 2.5, most references refer to $\mathbf{H}^\top$ as the parity-check matrix. In this thesis, we follow this notation. The encryption consists of computing the *syndrome* given by $\mathsf{c} = \mathbf{eH} \pmod{q-1}$. This ciphertext (or syndrome) is of length d, in contrast to the original length n, yielding a more compact representation. The underlying hard problem is the *syndrome decoding problem*, which is equivalent to the decoding problem in the McEliece setting.

To generate a public key, we proceed as in Section 3.2: select a polynomial $c(x)$ of degree d and a vector $\mathbf{a} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_q^n$. From these, construct a lattice basis and compute its Hermite Normal Form (HNF), resulting in a matrix of the form

$$\mathbf{B} = \begin{pmatrix} \mathbf{I}_{n-d} & \mathbf{G}' \\ 0 & (q-1)\mathbf{I}_d \end{pmatrix},$$

with high probability. The upper block $\mathbf{G} = \begin{pmatrix} \mathbf{I}_{n-d} & \mathbf{G}' \end{pmatrix}$ serves as the generator matrix of a $(q-1)$-ary linear code.

To obtain the corresponding parity-check matrix $\hat{\mathbf{H}}$, one computes the right kernel of $\mathbf{G}$ modulo $(q-1)$. The matrix $\hat{\mathbf{H}}$ is then transformed into *standard form* $\mathbf{H}$, in which the bottom $d \times d$ block is an identity matrix:

$$\mathbf{H} = \begin{pmatrix} -\mathbf{G}' \\ \mathbf{I}_d \end{pmatrix} \in \mathbb{Z}_{q-1}^{n\times d}.$$

Next we will show that we can directly compute $\mathbf{H}$ without computing $\mathbf{G}$.

4.2.1 Direct computation of the parity check matrix

This section gives an algorithm to construct the parity check matrix of the code directly. In this new construction, one does not have to consider the generator matrix $\mathbf{G}$ of the $(q-1)$-ary code. Instead, one can directly compute the parity check matrix $\hat{\mathbf{H}}$. We use the new formulation of the lattice introduced in Section 3.4.1.

Take q a prime and fix a generator g in $\mathbb{F}_q^*$. Let $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_d$ satisfy Condition 1. Now for all $1 \leq i \leq n$ and $1 \leq j \leq d$, let $a_{i,j} \in \mathbb{Z}_{q-1}$ be such that $(\beta_j - \alpha_i) = g^{a_{i,j}}$. Let $\hat{\mathbf{H}}$ be an $n \times d$ matrix with $(i,j)^{th}$ entry taken to be $a_{i,j}$ modulo $(q-1)$. So $\hat{\mathbf{H}} = [a_{i,j}] \in \mathbb{Z}_{q-1}^{n \times d}$.

Lemma 4.1. *$\hat{\mathbf{H}}$ is a parity check matrix for the $(q-1)$-ary code.*

Proof. A vector $\mathbf{u}$ is an element of the polynomial lattice if and only if

$$\prod_{i=1}^{n} (\beta_j - \alpha_i)^{u_i} = 1 \pmod{q}$$

for all $1 \leq j \leq d$. In other words, $\mathbf{u}$ is a codeword for the $(q-1)-$ary code. Using the generator g for $\mathbb{F}_q^*$, this is equivalent to

$$\prod_{i=1}^{n} (g^{a_{i,j}})^{u_i} = 1.$$

The exponents correspond to

$$\sum_{i=1}^{n} u_i a_{i,j} = 0 \pmod{q-1}.$$

This is equivalent to saying $\mathbf{u}\hat{\mathbf{H}} = \mathbf{0} \pmod{q-1}$ where $\mathbf{u}$ in row span of $\mathbf{G}$. That is, the row span of $\mathbf{G}$ modulo $(q-1)$ is characterised by $\mathbf{G}\hat{\mathbf{H}} = 0 \pmod{q-1}$. In other words, $\hat{\mathbf{H}}$ is the right kernel of $\mathbf{G}$ and so the rank of $\hat{\mathbf{H}}$ is d. □

Notice that this construction allows one to compute the parity check matrix directly from the secret keys $c(x)$ and $\mathbf{a} = \{\alpha_1, \ldots, \alpha_n\}$ without computing the generator matrix first. However, keeping this particular parity check matrix a secret is essential because, once revealed, an attacker can compute the secret keys as we now explain.

Why is it important to hide $\hat{\mathbf{H}}$? Suppose an attacker is given exactly the special parity check matrix $\hat{\mathbf{H}}$. There are less than q choices for the generator g of $\mathbb{F}_q^*$, so it is not hard to guess the generator. Consider the first two columns of $\hat{\mathbf{H}}$, namely $(a_{i,1}), (a_{i,2})$. For the correct guess of g and for each i, we have

$$g^{a_{i,1}} = (\beta_1 - \alpha_i) \text{ and } g^{a_{i,2}} = (\beta_2 - \alpha_i).$$

This means that,

$$g^{a_{i,1}} - g^{a_{i,2}} = (\beta_1 - \beta_2),$$

which is constant for all $1 \leq i \leq n$. This is a property easily detected by an attacker. Now, without loss of generality, one can take $\beta_1 = 0$, which means one learns β_2. Similarly, one can learn all the β_j's for $1 \leq j \leq d$ and hence obtain $c(x) = \prod_{j=1}^{d}(x - \beta_j)$. To determine the α_i's, consider the first column of $\hat{\mathbf{H}}$. For all $1 \leq i \leq n$, we already know $g^{a_{i,1}} = (\beta_1 - \alpha_i)$. Taking $\beta_1 = 0$, gives all the α_i's. Hence, we do not reveal this form of $\hat{\mathbf{H}}$.

Recall that in Condition 1 (in Chapter 3) we assumed that for a fixed i, the set $\{\beta_j - \alpha_i : j \in [d]\}$ generates $\mathbb{F}_q^*$. Now in terms of the generator g, this implies that for each i, the set $\{g^{a_{i,j}} : j \in [d]\}$ generates $\mathbb{F}_q^*$. Therefore, it follows that for a fixed i, the set of integers $\{a_{i,j} : 1 \leq j \leq d\}$ generates the additive group $\mathbb{Z}_{q-1}$. In other words, d entries in each row of $\hat{\mathbf{H}}$ span $\mathbb{Z}_{q-1}$. Given that $\hat{\mathbf{H}}$ has rank d, it follows that one can always convert $\hat{\mathbf{H}}$ into standard form. This is achieved by first computing the Hermite normal form (HNF) of $\hat{\mathbf{H}}$, treating its columns as vectors over $\mathbb{Z}$. The result is an upper triangular matrix. Then by permuting the rows of the HNF($\hat{\mathbf{H}}$) as needed, we obtain the standard form of the parity-check matrix which we write as $\mathbf{H} = \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix}$ where $\mathbf{H}' \in \mathbb{Z}_{q-1}^{n-d \times d}$ and $\mathbf{I}_d$ denotes the $d \times d$ identity matrix.

The standard form of the parity check matrix will be the public key, which hides the secret structure. We emphasise that the remarks made about hiding the original form of the parity check matrix do not lead to an attack, but they show there is a special structure in the dual code that may be relevant for cryptanalysis.

4.3 A KEM based on Niederreiter

The Key Encapsulation Mechanism (KEM) is now a standard construction in code-based cryptography, as exemplified by schemes like Classic McEliece [BCL$^+$17]. It was formally described and analysed by Bernstein and Persichetti [BP18].

In this section, we present the LLXY KEM construction in detail. Our scheme follows the Niederreiter framework that relies on the hardness of the syndrome decoding problem for security.

Key Generation

The key generation algorithm for the LLXY KEM takes as input the public parameters $n, q,$ and d and outputs a public pk and a private key sk as shown in Algorithm 1. To reduce the size of the public key, we set $pk = \mathbf{H}'$ where $\mathbf{H}'$ is the top part of the standard form of the parity check matrix $\mathbf{H}$ obtained in Lemma 4.1. The secret key contains the trapdoor information: the polynomial $c(x)$, n distinct elements in $\mathbb{F}_q$ defined as $\mathbf{a} = (\alpha_1, \ldots, \alpha_n)$ and a random value r sampled uniformly from a key space $\mathcal{K}$. The set $\mathcal{K}$ represent all equally likely possibilities for randomness r.

Algorithm 1 LLXY KEM Key Generation

Input: $n, q, d \in \mathbb{Z}$
Output: (sk, pk)

1: Choose distinct $\beta_j \leftarrow\$\ \mathbb{F}_q$ and compute $c(x) = \prod_{j=1}^{d}(x - \beta_j)$.
2: Choose distinct $\alpha_i \leftarrow\$\ \mathbb{F}_q$ and set $\mathbf{a} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_q^n$.
3: **if** $\forall\ i\ \{\beta_j - \alpha_i\ :\ j \in [d]\}$ generates $\mathbb{F}_q^*$, **then**
4: Proceed.
5: **else**
6: Go to Step 1.
7: **end if**
8: Choose $r \leftarrow\$\ \mathcal{K}$.
9: Compute $\hat{\mathbf{H}} \in \mathbb{Z}_{q-1}^{(n\times d)}$ using Lemma 4.1
10: Compute HNF($\hat{\mathbf{H}}$).
11: Permute rows of HNF($\hat{\mathbf{H}}$), if needed, until obtain $\mathbf{H} = \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix}$.
12: $pk \leftarrow \mathbf{H}'$
13: $sk \leftarrow (c(x), \mathbf{a}, r)$

Encapsulation

Given the public key $pk = \mathbf{H}'$, we first construct the full parity-check matrix $\mathbf{H}$ in systematic (or standard) form as $\mathbf{H} = \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix}$, where $\mathbf{I}_d$ denotes the $d \times d$ identity matrix. This construction ensures that $\mathbf{H}$ has full rank and allows for efficient syndrome computation during encryption.

To encapsulate a key, the sender begins by sampling a random error vector $\mathbf{e} \in \{-1, 0, 1\}^n$ such that its ℓ_1-norm satisfies $\|\mathbf{e}\|_1 \leq d - 1$. This restriction on the weight ensures that decoding remains feasible for the legitimate receiver.

The ciphertext is then computed as the syndrome of $\mathbf{e}$ under the parity-check matrix:

$$\mathsf{c} = \mathbf{e}\mathbf{H} \pmod{(q-1)},$$

which results in a vector of length d over $\mathbb{Z}_{q-1}$. The ciphertext effectively hides the error vector $\mathbf{e}$, which serves as the shared secret. The underlying hardness assumption is the *syndrome decoding problem*, which is believed to be computationally intractable for appropriate parameter choices.

The encapsulated key or the session key is derived using a cryptographic hash function:

$$K = \mathsf{H}(1, \mathbf{e}, \mathsf{c}). \tag{4.1}$$

Remark 4.1. *Unlike the approach by Li et al. [LLXY20], which employed binary error vectors from $\{0, 1\}^n$, our construction uses ternary error vectors from $\{-1, 0, 1\}^n$, following the design rationale proposed by Lapiha [Lap21]. The inclusion of negative entries changes the decoding algorithm, which we will discuss later.*

Decapsulation

The decapsulation algorithm makes use of the decoding algorithm for the polynomial lattice code. It operates as follows.

Given a ciphertext $\mathsf{c} \in \mathbb{Z}_{q-1}^d$, the algorithm first computes a vector $\mathbf{w} \in \mathbb{Z}_{q-1}^n$ (not necessarily of low weight or norm) such that

$$\mathsf{c} = \mathbf{w}\mathbf{H} \pmod{q-1},$$

where $\mathbf{H}$ is the parity-check matrix derived during key generation. Observe that the difference $\mathbf{w} - \mathbf{e}$ is a lattice vector, say $\mathbf{v} \in \mathcal{L}$, where $\mathbf{e}$ is the error vector used during encapsulation. The goal is to recover $\mathbf{e}$ by solving an instance of the Bounded Distance Decoding (BDD) problem, namely, to find the closest lattice vector $\mathbf{v}$ such that $\mathbf{w} = \mathbf{v} + \mathbf{e}$ and $\|\mathbf{e}\|_1 \leq d - 1$.

Using the trapdoor information (i.e., the secret key), the algorithm reconstructs the error vector $\mathbf{e}$ through the following computation. Writing $\mathbf{w} = (w_1, \ldots, w_n)$, we evaluate

$$\begin{aligned}\prod_{i=1}^{n}(x-\alpha_i)^{w_i} &= \prod_{i=1}^{n}(x-\alpha_i)^{v_i+e_i} = \prod_{i=1}^{n}(x-\alpha_i)^{v_i}\prod_{i=1}^{n}(x-\alpha_i)^{e_i} \\ &\equiv \prod_{i=1}^{n}(x-\alpha_i)^{e_i} \pmod{c(x)},\end{aligned} \tag{4.2}$$

where $c(x)$ is the polynomial of degree d. Thus, the left-hand side of the equation yields (modulo $c(x)$) the encoded form of the error vector $\mathbf{e}$, which can be decoded using continued fractions and polynomial factorisation, provided that $\|\mathbf{e}\|_1 \leq d-1$.

If decoding is successful, the algorithm reconstructs $\mathbf{e}$ and derives the session key as $K = \mathsf{H}(1, \mathbf{e}, \mathsf{c})$. If decoding fails due to an invalid ciphertext, the algorithm does not output a failure symbol. Instead, to enable a tight CCA-security proof, it performs *implicit rejection*: the algorithm deterministically produces a pseudorandom key K computed as $K = \mathsf{H}(0, r, \mathsf{c})$, where r is an independent secret included in the KEM private key.

Implicit Rejection. Implicit rejection refers to the procedure in which invalid ciphertexts result in pseudorandom outputs rather than explicit failure signals. This approach was first introduced by Persichetti [Per12], and a systematic comparison between implicit and explicit rejection was later carried out by Hofheinz et al. [HHK17]. Bernstein and Persichetti [BP18] provide an in-depth discussion of the technique and recommend its use in KEM constructions.

Algorithm 2 LLXY KEM Decapsulation

Input: $\mathsf{c}, sk = (c(x), \mathbf{a}, r)$
Output: K

1: Compute $\mathbf{w} \in \mathbb{Z}_{q-1}^{n}$ such that $\mathsf{c} = \mathbf{wH} \in \mathbb{Z}_{q-1}^{d}$
2: **if** $\mathbf{e} \leftarrow$ **Decode**$(\mathbf{w}, c(x), \mathbf{a})$ **then**
3: **return** $K = \mathsf{H}(1, \mathbf{e}, \mathsf{c})$
4: **else**
5: **return** $K = \mathsf{H}(0, r, \mathsf{c})$
6: **end if**

4.3.1 How the Decoding Works for Ternary Errors

In the case where the error vector $\mathbf{e}$ is binary, (i.e., $\mathbf{e} \in \{0,1\}^n$) the decoding algorithm **Decode** operates by factoring the associated polynomial. This method is already described in the original LLXY paper [LLXY20], which shows that the error support can be recovered

by factoring the polynomial

$$\prod_{i=1}^{n}(x-\alpha_i)^{w_i} \pmod{c(x)},$$

as each $(x-\alpha_i)$ with $e_i = 1$ appears linearly in the exponent.

However, this factorisation method no longer suffices when the error vector includes both positive and negative entries, as is the case in our scheme, where $\mathbf{e} \in \{-1, 0, 1\}^n$. In such cases, we adopt the decoding method for arbitrary discrete errors introduced by Ducas and Pierrot [DP19], which is based on continued fractions.

Recall from Equation (4.2) that for any error vector $\mathbf{e} = (e_1, \ldots, e_n)$, we have

$$\prod_{i=1}^{n}(x-\alpha_i)^{w_i} \equiv \prod_{i=1}^{n}(x-\alpha_i)^{e_i} \pmod{c(x)}.$$

When the error vector contains both positive and negative components, the product on the right-hand side can be naturally rewritten as a rational function

$$\begin{aligned}\prod_{i=1}^{n}(x-\alpha_i)^{w_i} &\equiv \prod_{i=1}^{n}(x-\alpha_i)^{e_i} \pmod{c(x)} \\ &\equiv \frac{\prod_{i,\, e_i>0}(x-\alpha_i)^{e_i}}{\prod_{i,\, e_i<0}(x-\alpha_i)^{-e_i}} = \frac{U(x)}{V(x)} \pmod{c(x)},\end{aligned}$$

where $U(x)$ and $V(x)$ are polynomials in $\mathbb{F}_q[x]$ determined by the positive and negative entries of $\mathbf{e}$ respectively, and $\deg(U(x)) + \deg(V(x)) = \|\mathbf{e}\|_1$.

Let us define $X(x) = \prod_{i=1}^{n}(x-\alpha_i)^{w_i} \mod c(x)$. Then the above congruence becomes an equation in $\mathbb{F}_q[x]$,

$$V(x) \cdot X(x) = U(x) + k(x) \cdot c(x),$$

for some polynomial $k(x) \in \mathbb{F}_q[x]$. Rearranging terms yields the rational approximation

$$\frac{X(x)}{c(x)} - \frac{k(x)}{V(x)} = \frac{U(x)}{c(x)V(x)}. \tag{4.3}$$

The task of recovering U and V can now be interpreted as solving a Diophantine approximation problem over the function field $\mathbb{F}_q(x)$. Specifically, we aim to approximate the rational function $X(x)/c(x)$ by another rational function $k(x)/V(x)$ such that the approximation error, now measured in terms of degree, is small.

Classical results on Diophantine approximation extend to the polynomial setting. In particular, Theorem 3 of [Wal18] gives a power series analogue of Legendre's Theorem: if $x \in \mathbb{F}_q((1/T)) \setminus \mathbb{F}_q(T)$ and

$$\left| x - \frac{P}{Q} \right| < \frac{1}{|Q|^2}, \tag{4.4}$$

then P/Q is a convergent in the continued fraction expansion of x. In the polynomial setting, the absolute value is defined by

$$\left| \frac{P}{Q} \right| = e^{\deg(P) - \deg(Q)},$$

so the above Equation (4.4) becomes a degree inequality as follows,

$$\deg\left(x - \frac{P}{Q} \right) < -\deg(Q^2).$$

Applying this to our case, and observing that the error term in Equation (4.3) is $\frac{U}{c(x)V}$, we compute its degree as:

$$\deg\left(\frac{U}{c(x)V} \right) = \deg(U) - \deg(c(x)V) = -\left[\deg(c(x)V) - \deg(U)\right].$$

Therefore, if

$$\deg(c(x)V) - \deg(U) > \deg(V^2),$$

or, more precisely, if

$$\deg(c(x)) > \deg(V) + \deg(U),$$

then the approximation $k(x)/V(x)$ must be a convergent of the continued fraction expansion of $X(x)/c(x)$. It follows that the Extended Euclidean Algorithm (EEA) (or continued fraction algorithm) applied to the pair $(X(x), c(x))$ will output a list of rational approximations, among which the correct triple (U, k, V) appears.

Finally, we note that this degree condition is satisfied in our setting. Since the ℓ_1-norm of the error vector is $\|\mathbf{e}\|_1 < \deg(c(x))$, and both U and V satisfy the following equation

$$\deg(U) + \deg(V) = \|\mathbf{e}\|_1 < \deg(c(x)),$$

the above condition on degrees is true for our case.

The decoding algorithm proceeds by computing all candidate triples (U_i, k_i, V_i) such that the identity $V_i X = U_i + k_i \cdot c(x)$ holds and $\deg(U_i) + \deg(V_i) < d$. These candidates are computed using the extended Euclidean algorithm (EEA), denoted by the function **EEA**

in Algorithm 3. For each candidate triple, the algorithm checks whether both $U(x)$ and $V(x)$ factor completely into linear terms of the form $(x - \alpha_i)$ over the field $\mathbb{F}_q$.

If none of the candidate pairs (U, V) satisfy this factorisation property, then the decoding algorithm concludes that no valid error vector can be recovered from the input ciphertext $\mathbf{c}$, and decoding fails. This may occur, when $\mathbf{c}$ is not a valid syndrome, such as when it is randomly chosen.

Conversely, if more than one distinct pair (U, V) satisfies the factorisation condition, then the algorithm is unable to uniquely determine the correct error vector $\mathbf{e}$. Decoding (and hence decapsulation) fails in this case as well.

We summarise the full decoding procedure in Algorithm 3, which integrates these steps: computing rational approximations via EEA, verifying the correct factorisation of the candidates, and recovering the error vector $\mathbf{e}$ when a unique valid candidate is found.

Algorithm 3 LLXY KEM Decode

Input: $\mathbf{w}, c(x), \mathbf{a}$
Output: $\mathbf{e}$ or $\perp$

Compute $X \equiv \prod_{i=1}^{n}(x - \alpha_i)^{w_i} \pmod{c(x)}$
candidates $\leftarrow \emptyset$
$S = (k_i, U_i, V_i) \leftarrow$ **EEA**$(X, c(x))$
while $i = 1, \ldots, |S|$ **do**
    if $U_i \equiv \prod_{i=1}^{n}(x - \alpha_i)^{e_i}$ and $V_i \equiv \prod_{i=1}^{n}(x - \alpha_i)^{f_i}$ **then**
        candidates $\leftarrow$ candidates $\cup \{(e, f)\}$
    end if
end while
if $|$candidates$| \neq 1$ **then**
    return $\perp$
else
    return $\mathbf{e} = e - f$ where candidates $= \{(e, f)\}$
end if

4.3.2 Avoiding decapsulation failures.

We now explain how to choose parameters in order to minimize the risk of decoding or decapsulation failure caused by non-uniqueness in the recovered error vector. Specifically, we focus on the case where two or more distinct continued fraction convergents, say (U, V) and (U', V'), both satisfy the condition that their corresponding rational functions $\frac{U}{V}$ and $\frac{U'}{V'}$, factor completely over the factor base (i.e., all roots are among the α_i). If this occurs for a correctly formed ciphertext, it means both rational functions represent valid error vectors, and the decoding algorithm cannot distinguish which one is correct.

Since both pairs satisfy

$$X \equiv \frac{U}{V} \pmod{c(x)} \quad \text{and} \quad X \equiv \frac{U'}{V'} \pmod{c(x)},$$

it follows that

$$UV' - U'V \equiv 0 \pmod{c(x)},$$

so $c(x)$ divides the polynomial $UV' - U'V$. Note that this expression involves only polynomials built from factors of the form $(x - \alpha_i)$, which implies that $UV' - U'V$ factors completely over the factor base. Consequently, the existence of such a relation implies the presence of a surprisingly short non-zero vector in the underlying lattice.

To understand the impact, let (U, V) correspond to an error vector $\mathbf{e}_1 \in \{-1, 0, 1\}^n$, and let (U', V') correspond to another error vector $\mathbf{e}_2 \in \{-1, 0, 1\}^n$. Since both give rise to the same ciphertext, we must have $\mathbf{e}_1\mathbf{H} = \mathbf{e}_2\mathbf{H}$, which implies that their difference $\mathbf{e}_1 - \mathbf{e}_2 \in \{-2, -1, 0, 1, 2\}^n$ lies in the lattice. We also know that $\|\mathbf{e}_1\|_1 < d$ and $\|\mathbf{e}_2\|_1 < d$, so the difference vector $\mathbf{e}_1 - \mathbf{e}_2$ is non-zero and has small norm. Specifically, its ℓ_1-norm satisfies $\|\mathbf{e}_1 - \mathbf{e}_2\|_1 \leq 2d$. A direct application of the standard norm inequality $\|\mathbf{v}\|_2 \leq \|\mathbf{v}\|_1$ for $\mathbf{v} \in \mathbb{R}^n$ implies that $\|\mathbf{e}_1 - \mathbf{e}_2\|_2 \leq 2d$. However, we can obtain a significantly tighter bound by using the sparsity of the vector $\mathbf{e}_1 - \mathbf{e}_2$. Since $\mathbf{e}_1 - \mathbf{e}_2$ contains at most d non-zero entries each with a maximum absolute value of 2, its Euclidean norm is bounded by $\|\mathbf{e}_1 - \mathbf{e}_2\|_2 \leq \sqrt{4d} = 2\sqrt{d}$.

Based on Lemma 3.1 and Remark 3.1, we use the Gaussian heuristic as a lower bound estimate for the length of the shortest vector. According to this, the shortest non-zero vector in $\mathcal{L}_{a,c(x)}$ has norm approximately $\sqrt{\frac{n}{2\pi e}}\,(q-1)^{d/n}$ (see Equation (2.1)). If the bound $2\sqrt{d}$ is significantly smaller than this predicted shortest length, then it is heuristically unlikely that the lattice contains such a short non-zero vector. Hence, by appropriately choosing the parameters n, d, and q, one can make it overwhelmingly likely that the lattice does not contain any such non-zero vector of norm less than $2\sqrt{d}$. In this case, decapsulation failures due to non-unique decoding are provably ruled out, as no two distinct ternary error vectors map to the same syndrome under $\mathbf{H}$.

As evidence, Table 4.1 provides a comparison between the actual length of the shortest vector in LLXY lattices and the corresponding estimates given by the Gaussian heuristic. The value for λ_1 shown in the table represents the minimum observed norm over 100 randomly generated LLXY lattices for each parameter set.

One can observe that the shortest vector is consistently at least as long as the value predicted by the Gaussian heuristic. This empirical behaviour supports the assumption that

the Gaussian heuristic provides a reliable lower bound on the length of the shortest non-zero vector in LLXY lattices.

n	d	q	Actual λ_1	Gaussian heuristic estimate
50	8	59	3.3166	3.2854
60	10	71	3.8730	3.8140
70	12	83	4.3589	4.3181

Table 4.1: Comparison of length of shortest vector with Gaussian Heuristic estimate.

Table 4.2 presents the parameter sets we propose for the LLXY KEM (see Section 5.4.2) and compares the Gaussian heuristic estimate for the shortest non-zero lattice vector with the ℓ_2-norm bound of the difference vector $\mathbf{e}_1 - \mathbf{e}_2$ that could arise in the case of non-unique decoding. The results show that the Gaussian heuristic lower bound is substantially larger than the norm bound $2\sqrt{d}$ of any vector coming from non-unique decoding. This provides strong confidence that for these parameter sets, the LLXY lattice does not contain such short non-zero vectors, and therefore decoding (and decapsulation) errors due to non-uniqueness do not exist.

n	d	q	Gaussian heuristic estimate	$2\sqrt{d}$
898	260	1409	59.17	32.25
1390	386	2209	76.55	39.29
1840	520	2371	93.33	45.61

Table 4.2: Comparison of estimated length of the shortest vector with $\|\mathbf{e}_1 - \mathbf{e}_2\|_2$ bound.

In case of confusion, we re-state what this means. We are **not** claiming that, for a given public key, there is a low probability of non-unique decoding, averaged over the ciphertexts. The claim is much stronger than this. We have shown that any non-unique decoding implies the existence of a vector of length much smaller than the predicted shortest vector according to the Gaussian heuristic, and we have provided evidence that our lattices behave consistently with the Gaussian heuristic. Therefore, we assert that decapsulation works perfectly for the parameters we propose.

4.4 Security Proof

In this section we outline the standard CCA security proof for the Niederreiter KEM. It is well-known that the security of Niederreiter and McEliece for the same code are equivalent

[LDW94, BLP08]. We assume that decapsulation always works perfectly (i.e., there are no decryption/decoding errors), which we believe to be the case based on our experimental evidence on Section 4.3.2.

We refer to [BP18, JZM21, BHH+19] for the general definitions and the security proof in QROM. The security proof of the LLXY KEM is based on the hardness of syndrome decoding for the lattice code. This implicitly includes the problem of determining the secret key from the public key.

Definition 4.1. *Given a parity check matrix* $\mathbf{H} \in \mathbb{Z}_{q-1}^{n \times d}$ *and a syndrome* $\mathsf{c} \in \mathbb{Z}_{q-1}^{d}$*, the* LLXY syndrome decoding problem *is to find* $\mathbf{e} \in \{-1, 0, 1\}^n$ *such that* $\|\mathbf{e}\|_1 \leq d - 1$ *and* $\mathsf{c} = \mathbf{eH}$.

The LLXY syndrome decoding problem can be interpreted as a bounded distance decoding problem in a linear code over $\mathbb{Z}_{q-1}$. When $\mathbf{H}$ is viewed as a uniformly random parity-check matrix, this problem is analogous to the classical syndrome decoding problem, which is NP-hard [BMVT78]. We therefore assume that no polynomial-time algorithm can solve Definition 4.1 with non-negligible probability in the parameter regime of interest.

We recall the standard definition of IND-CCA security for a KEM (Definition 12, [BHH+19]).

Definition 4.2. *Let* K *be a KEM. The security experiment* $\mathsf{Expt}_{\mathsf{K}}^{\mathsf{IND-CCA}}(\mathcal{A})$ *is defined in Figure 4.1 for an adversary* $\mathcal{A}$ *against* K*, given access to a (quantum-accessible) random oracle* H *and a classical decapsulation oracle* D*.*

We define the advantage of a classical (respectively quantum) adversary $\mathcal{A}$ *against a KEM* K *in the classical (respectively quantum-accessible) random oracle model as*

$$\mathsf{Adv}_{\mathsf{K}}^{\mathsf{IND-CCA}}(\mathcal{A}) = \left| \Pr[\mathsf{Expt}_{\mathsf{K}}^{\mathsf{IND-CCA}}(\mathcal{A}) = 1] - \frac{1}{2} \right|. \tag{4.5}$$

The IND-CCA security of the LLXY KEM follows from standard results (see Bernstein and Persichetti [BP18] and Bindel, Hamburg, Hövelmanns, Hülsing, and Persichetti [BHH+19]), assuming the passive security of the trapdoor function (which is the syndrome decoding problem for the code). Note that CCA security in classic ROM, [BP18] requires a perfectly correct decapsulation algorithm, which was the case for the error vectors proposed in the original LLXY paper and Lapiha's thesis, and we claim will be the case for our variant as well based on our previous discussions on Remark 3.1 and experimental evidence from Section 4.3.2.

The standard KEM transfroms make explicit the cryptographic assumptions and correctness requirements imposed on the underlying public-key encryption scheme, and we

$\underline{\mathsf{Expt}^{\mathsf{IND-CCA}}_{\mathsf{K}}(\mathcal{A})}$:
1: $\mathsf{H} \leftarrow\!\$\ \mathcal{H}$
2: $(pk, sk) \leftarrow \mathsf{KeyGen}()$
3: $(c^*, k_0^*) \leftarrow \mathsf{Encap}(pk)$
4: $k_1^* \leftarrow\!\$\ K$
5: $b \leftarrow\!\$\ \{0,1\}$
6: $b' \leftarrow \mathcal{A}^{\mathsf{H},D}(pk, c^*, k_b^*)$
7: **return** $[b = b']$

$\underline{\text{Decapsulation oracle } D(\mathsf{c}) :}$
1: **if** $\mathsf{c} = c^*$ **then**
2: **return** $\perp$
3: **end if**
4: **return** $\mathsf{Decap}(sk, \mathsf{c})$

Figure 4.1: IND-CCA security experiment in the (Q)ROM against an adversary $\mathcal{A}$

explain how these requirements are satisfied by the LLXY construction. The KEM transform takes as input OW-CPA secure deterministic public-key encryption and output an IND-CCA secure KEM. In the LLXY setting, we instantiates such a deterministic PKE as follows: The key generation algorithm output $\mathbf{H}' \in \mathbb{Z}_{q-1}^{(n-d)\times d}$ using Lemma 4.1 (similar to Algorithm 1) as the public key and trapdoor information $(c(x), \mathbf{a})$ as the secret key. The plaintext space $\mathcal{P}$ consists of ternary vectors such that $\mathcal{P} = \{\mathbf{e} \in \{-1,0,1\}^n \ : \ \|\mathbf{e}\|_1 \leq d-1\}$. Encryption algorithm returns a ciphertext $\mathsf{c} \in \mathbb{Z}_{q-1}^d$ such that $\mathsf{c} = \mathbf{e}\mathbf{H} \pmod{(q-1)}$ where $\mathbf{H} = \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix}$. Upon receiving the c, the decryption algorithm (Decode Algorithm 3) either recovers $\mathbf{e}$ using the secret key $(c(x), \mathbf{a})$ or output the failure symbol $\perp$. The security of the KEM transform relies on the one-wayness of the underlying deterministic PKE. Now we refer to the definition of the OW-CPA security (Definiton 2.2) and prove that the OW-CPA security of the LLXY deterministic PKE directly reduces to the hardness of the LLXY syndrome decoding problem.

Lemma 4.2. *If there exists a PPT adversary $\mathcal{A}$ that breaks the OW-CPA security of the LLXY deterministic PKE scheme with non-negligible probability, then there exists a PPT algorithm that solves the LLXY Syndrome Decoding Problem (LLXY-SDP) with non-negligible probability.*

Proof. Assume there exists a probabilistic polynomial-time adversary $\mathcal{A}$ that breaks the OW-CPA security of the LLXY deterministic public-key encryption scheme with non-negligible probability $\varepsilon(\lambda)$. That is, given a public key $pk = \mathbf{H}'$ and a ciphertext $\mathsf{c} = \mathbf{e}\mathbf{H}$, where $\mathbf{H} = \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix}$ is distributed according to the key generation algorithm and

$\mathbf{e} \in \{-1, 0, 1\}^n$ satisfies $\|\mathbf{e}\|_1 \leq d - 1$, the adversary outputs the plaintext $\mathbf{e}$ with probability at least $\varepsilon(\lambda)$.

We construct a probabilistic polynomial-time algorithm $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine to solve the LLXY Syndrome Decoding Problem. Given an instance $(\mathbf{H}, \mathsf{c})$ of the LLXY syndrome decoding problem, where $\mathbf{H} = \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix}$ and $\mathsf{c} = \mathbf{e}\mathbf{H}$ for some $\mathbf{e} \in \{-1, 0, 1\}^n$ with $\|\mathbf{e}\|_1 \leq d - 1$, the algorithm $\mathcal{B}$ sets $pk := \mathbf{H}'$ and forwards the pair (pk, c) to $\mathcal{A}$. By assumption, since $\mathcal{A}$ breaks OW-CPA security of the LLXY PKE, it outputs the corresponding plaintext vector $\mathbf{e}$ with probability at least $\varepsilon(\lambda)$. Thus, Algorithm $\mathcal{B}$ outputs this vector as its solution to the syndrome decoding instance.

Therefore,

$$\Pr[\mathcal{B} \text{ solves LLXY-SDP}] = \Pr[\mathcal{A} \text{ breaks OW-CPA}] \geq \varepsilon(\lambda),$$

which is non-negligible. Since $\mathcal{B}$ runs in polynomial time whenever $\mathcal{A}$ does, this yields a probabilistic polynomial-time algorithm that solves the LLXY Syndrome Decoding Problem with non-negligible probability. Hence, under the hardness assumption of the LLXY Syndrome Decoding Problem, the LLXY deterministic PKE scheme is OW-CPA secure. □

Another requirement for the KEM transform is the perfect correctness of the underlying PKE. In our setting, this corresponds to the assumption that the decoding algorithm always successfully recovers the unique error vector $\mathbf{e} \in \{-1, 0, 1\}^n$ for a valid syndrome, where $\|\mathbf{e}\|_1 \leq d - 1$. In Section 4.3.2 we support this argument through experimental results. We show that any decoding failure would imply the existence of an unusually short non-zero lattice vector, contradicting both the Gaussian heuristic estimate on the length of the shortest vector (based on Remark 3.1) and experimental evidence (Section 4.3.2). Hence for the selected parameters we assume the perfect correctness of the underlying PKE.

The KEM transform additionally requires that the underlying deterministic encryption function be ε-injective, except with negligible probability.

Definition 4.3. *A deterministic PKE is said to be ε-injective if*

$$\Pr[\mathsf{Enc}(pk, \mathsf{m}) = \mathsf{Enc}(pk, \mathsf{m}') \textit{ for some } \mathsf{m} \neq \mathsf{m}'] \leq \varepsilon$$

over the randomness of key generation.

In the LLXY scheme, injectivity follows from correctness and uniqueness of decoding. If $\mathsf{c}_1 = \mathsf{c}_2$ for some $\mathbf{e}_1 \neq \mathbf{e}_2$, we have

$$\mathbf{e}_1\mathbf{H} = \mathbf{e}_2\mathbf{H} \pmod{(q-1)}.$$

Then $(\mathbf{e}_1 - \mathbf{e}_2)$ lies in the associated LLXY lattice. As shown in Section 4.3.2, any such non-zero vector would have Euclidean norm at most $2\sqrt{d}$, which is significantly smaller than the expected shortest vector length predicted by the Gaussian heuristic. Empirical evidence supports that no such vectors exist for the proposed parameters. Hence, we heuristically expect encryption is injective and $\varepsilon = 0$.

Theorem 14.3 of [BP18] gives a tight bound for CCA security in the classical ROM, and Theorem 2 of [BHH$^+$19] gives a bound for the CCA advantage in the quantum ROM. A paper by Chen et al. [CWC$^+$24] discusses tighter proofs for PKE-to-KEM transformation in the QROM model. They convert OW-CPA secure PKEs into KEMs with IND-1CCA security, a variant of IND-CCA security where only a single decapsulation query is allowed.

For the QROM proof, we will follow the Theorem 2 of [BHH$^+$19]. The $U^{\not\perp}$ transform described in their paper, is essentially the SimpleKEM transform defined in [BP18]. We show that our scheme can be viewed as a special case of the $U^{\not\perp}$ transform. The $U^{\not\perp}$ transform turns an OW-CPA secure public-key encryption scheme into an IND-CCA secure key-encapsulation mechanism using a pseudo-random function (PRF) F and a hash function H. The notation $\not\perp$ in $U^{\not\perp}$ signifies that, upon receiving an invalid ciphertext, the decapsulation returns a pseudo-random key ("implicit rejection") instead of an explicit rejection symbol $\perp$ [HHK17]. The key generation algorithm is identical to Algorithm 1 (in Section 4.3) that generates the secret key together with the pseudo-random key (prfk) used in implicit rejection and the public keys. The encapsulation algorithm computes $\mathsf{H}(m, \mathsf{c})$ based on a randomly generated message m encrypted with the public key, where m aligns with the error vector $\mathbf{e}$ in LLXY encapsulation.

A notable difference between the $U^{\not\perp}$ transform and our decapsulation algorithm (Algorithm 2) is that in the $U^{\not\perp}$ transformation, Decapsulation checks three cases: the first two address decryption failures, while the last case covers successful decryption. An invalid ciphertext arises if decryption fails or if re-encrypting the decrypted message does not reproduce the original ciphertext. In these cases, $U^{\not\perp}$-KEM returns a deterministic output $\mathsf{F}(\text{prfk}, \mathsf{c})$. If no such failure occurs, it returns $\mathsf{H}(m, \mathsf{c})$. In contrast, Algorithm 2 reduces this to two cases, as re-encrypting a decrypted error vector will always yield the original ciphertext. If decoding fails (i.e., if Decode Algorithm 3 returns the abort symbol $\not\perp$), our decapsulation scheme outputs a pseudo-random value $\mathsf{H}(0, r, \mathsf{c})$. Otherwise, it returns the successful decapsulation result $\mathsf{H}(1, \mathbf{e}, \mathsf{c})$. Therefore, the LLXY KEM is a

$U^{\not\perp}$-transformed KEM with perfect decapsulation, making the security proofs for the two constructions equivalent.

Theorem 4.1. *Assuming perfect correctness and injectivity of the underlying deterministic PKE, and assuming hardness of LLXY-SDP, the LLXY KEM is IND-CCA secure in the QROM in the random oracle model.*

The proof follows Theorem 2 of [BHH+19]. To quote their theorem:

Let $\mathsf{H} : \mathcal{M} \times \mathcal{C} \to \mathcal{K}$ be a quantum-accessible random oracle and $\mathsf{F} : \mathcal{K}_F \times \mathcal{C} \to \mathcal{K}$ be a PRF. Let P be an ϵ-injective dPKE which is independent of H. Let $\mathcal{A}$ be an IND-CCA adversary against the KEM $U^{\not\perp}(\mathsf{P}, \mathsf{F})$, and suppose that $\mathcal{A}$ makes at most q_{dec} decryption queries. Then we can construct three adversaries running in about the same time and resources as $\mathcal{A}$:

- an OW-CPA-adversary $\mathcal{B}_1$ against P
- a FFC-adversary $\mathcal{B}_2$ ("failure finding") against P, returning a list of at most q_{dec} ciphertexts
- a PRF-adversary $\mathcal{B}_3$ against F

such that

$$\mathsf{Adv}^{\mathsf{IND-CCA}}_{U^{\not\perp}(\mathsf{P})}(\mathcal{A}) \leq 2\sqrt{\mathsf{Adv}^{\mathsf{OW-CPA}}_{\mathsf{P}}(\mathcal{B}_1)} + 2\mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}}(\mathcal{B}_3) + \mathsf{Adv}^{\mathsf{FFC}}_{\mathsf{P}}(\mathcal{B}_2) + \epsilon \tag{4.6}$$

A deterministic public-key encryption scheme (dPKE) is defined the same way as an encryption scheme (see Figure 2.1), except that the encryption is a deterministic algorithm. A dPKE $\mathsf{P} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is called ϵ-injective if the encryption function is injective with high probability. As discussed earlier, the LLXY deterministic PKE is injective: distinct $\mathbf{e}$ produce distinct ciphertexts $\mathsf{c} = \mathbf{eH} \pmod{q-1}$. Under this claim, we obtain an injective dPKE with $\epsilon = 0$.

The PRF in the $U^{\not\perp}$ transform is used for implicit rejection, returning $\mathsf{F}(\text{prfk}, \mathsf{c})$ in case of an invalid ciphertext using a secret prfk. In the LLXY KEM, $\mathsf{F}(\text{prfk}, \mathsf{c})$ is implemented as $\mathsf{H}(0, r, \mathsf{c})$. In particular, Step 5 in our Algorithm 2 serves this purpose by outputting a deterministic value $K = \mathsf{H}(0, r, \mathsf{c})$ with the secret r when the decoding does not succeed. For valid ciphertexts, the decapsulation algorithm instead outputs $\mathsf{H}(1, \mathbf{e}, \mathsf{c})$, providing domain separation between successful and unsuccessful decapsulation.

A new failure-finding experiment, *find-failing-ciphertext* (FFC) captures the probability that the adversary can find valid ciphertexts that cause a decryption failure. In Section 4.3 we showed that we have a correct PKE without decapsulation failures and therefore, the

FFC advantage is zero. Consequently, when applying inequality (4.6) to the LLXY scheme, the bound reduces to the OW-CPA advantage term together with the PRF-security term. As noted by [JZM21], the quadratic loss in the QROM reduction may be unavoidable. We refer the reader to [BHH$^+$19] for the complete proof of the transform.

Now the proof of Theorem 4.1 directly follows.

Proof. The LLXY KEM can be viewed as an instance of the $U^{\not\perp}$ transform of [BHH$^+$19], instantiated with the LLXY deterministic public-key encryption scheme P. As discussed above, the underlying dPKE is assumed to be injective, and the implicit rejection mechanism is implemented in the random oracle model through the deterministic output $\mathsf{H}(0, r, \mathsf{c})$ using the secret value r. By Theorem 2 of [BHH$^+$19], any adversary that breaks the IND-CCA security of the resulting KEM in the QROM can be transformed into an adversary that either:

- breaks the OW-CPA security of the underlying deterministic PKE P,
- breaks the PRF used for implicit rejection, or
- finds a ciphertext causing a decapsulation failure.

Under the perfect correctness assumption for the LLXY decoding algorithm, decapsulation failures do not occur, and therefore the failure-finding advantage is zero. Moreover, in the random oracle model, the value $\mathsf{H}(0, r, \mathsf{c})$ with randomness r serves the role of the PRF required by the $U^{\not\perp}$ transform. [BHH$^+$19] claims that breaking the PRF harms security if and only if implicit rejection is more secure than explicit rejection and they give a theorem (Theorem 3 in [BHH$^+$19]) to show that the transform $U^{\perp}$ (with explicit rejection) never yields KEMs that are more secure than KEMs constructed via $U^{\not\perp}$ (with implicit rejection). Consequently, the IND-CCA security of the LLXY KEM reduces to the OW-CPA security of the underlying LLXY deterministic PKE.

Finally, Lemma 4.2 shows that any adversary breaking the OW-CPA security of the LLXY deterministic PKE yields an algorithm solving the LLXY syndrome decoding problem with non-negligible probability. Therefore, assuming the hardness of the LLXY syndrome decoding problem together with the perfect correctness and injectivity of LLXY deterministic PKE discussed above, the LLXY KEM is IND-CCA secure in the QROM. □

4.5 Summary

In this chapter, we reformulated the LLXY scheme as a Key Encapsulation Mechanism (KEM) based on the Niederreiter cryptosystem. This formulation enables the use of the

parity-check matrix, resulting in significantly smaller ciphertexts compared to the original encryption scheme. We also showed how to compute the parity-check matrix directly, without first computing a generator matrix of the $(q-1)$-ary code, and explained decoding for ternary errors while avoiding decapsulation failures. Despite these structural changes, the scheme maintains standard levels of cryptographic security.

We further demonstrated how to achieve Chosen Ciphertext Attack (CCA) security using standard techniques, showing that it follows from the OW-CPA security of the syndrome decoding problem.

In the next chapter, we provide a detailed security analysis of the scheme, considering state-of-the-art lattice and decoding attacks from the literature.

Chapter 5

Cryptanalysis of the LLXY KEM

Contents

This chapter provides a detailed security analysis of the scheme, incorporating the state-of-the-art known lattice-based and decoding attacks from the literature. In the last section we propose parameters for the LLXY KEM to avoid these attacks.

Recall that Lapiha [Lap21] provided decoding attacks for the parameters suggested by Li et al. [LLXY17] and [LLXY20] revised the parameters and gave the BKZ cost for those new parameters. In both works, Li et al. have only considered the embedding technique

(primal attack). However, our analysis on the lattice attack using the lattice estimator tool (a.k.a the LWE estimator) [APS15] suggests that dual/hybrid attacks work better than the embedding attack to solve the LLXY scheme.

In Table 5.1 we give the revised BKZ costs claimed by the LLXY [LLXY20], and the LWE estimator results for the LLXY parameters with errors in $\{0,1\}^n$ and a lower bound for the decoding cost based on two-list near neighbour methods [BGLS19]. This underscores why we have to reconsider the parameters for the LLXY scheme for required post-quantum security level.

n	d	q	$\log_2$(LLXY BKZ cost)	$\log_2$(LWE estimator cost)	lower bound $\log_2$(NN-ISD cost)
285	41	2819	80	42	80
500	43	29599	128	40	103
729	42	152003	180	38	115

Table 5.1: Cost of lattice and decoding attacks for LLXY [LLXY20] parameters.

5.1 Lattice Attacks

As we have mentioned, breaking an LLXY ciphertext is solving a BDD instance in an $(q-1)$-ary lattice. There are many different lattice attacks in the literature, such as reducing to unique-SVP, dual attacks, hybrid attacks, etc. To analyse all these different attack strategies in the context of LLXY would be a substantial task. Fortunately, we can save time by using the lattice estimator [APS15], which already contains the most common and commonly performant attack strategies for lattice problems. We first show how to convert an LLXY ciphertext into an LWE instance (see Definition 2.13) so that it can be input to the lattice estimator.

Convert LLXY ciphertext to an LWE instance.

Consider a parity check matrix $\mathbf{H} = \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix} \in \mathbb{Z}_{q-1}^{n\times d}$ in standard form and an error vector $\mathbf{e} \in \{-1,0,1\}^n$ that satisfies the equation

$$\mathsf{c} = \mathbf{e}\begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix} \quad \text{where} \quad \|\mathbf{e}\|_1 < d. \tag{5.1}$$

This is an instance of the Inhomogeneous Short Integer Solution (ISIS) problem, and it can be reduced to an LWE instance by splitting the error vector as $\mathbf{e} = \begin{pmatrix} \mathbf{e}' & \mathbf{e}'' \end{pmatrix}$ such that

$$\mathsf{c} = \mathbf{e}'\mathbf{H}' + \mathbf{e}''. \tag{5.2}$$

Here $\mathbf{e}'$ is the LWE secret of length $(n-d)$ and $\mathbf{e}''$ is the LWE error of length d. Hence, this is a special case of an LWE instance $(\mathbf{H}', \mathsf{c})$, where the secret and the error are taken from a sparse low norm distribution with ℓ_1-norm bounded by $w < d$. Each vector is chosen uniformly at random among all vectors of weight at most w, and nonzero entries take values ± 1 with equal probability.

It is important to note that the secret vector in a standard LWE instance (Defintion 2.13) is uniquely determined with high probability, provided the number of samples m is sufficiently large (typically $m = \Omega(n)$) and the error magnitude is sufficiently small relative to the modulus q. Accoding to our parameter choices, $(n-d) > d$ and hence we have fewer samples than the secret dimension. Therefore, in the standard setting this would be an underdetermined LWE instance. However in the LLXY setting the uniqueness of the secret vector is guaranteed by the sparcity of the vector $\mathbf{e}$. Note that the number of possible $\mathbf{e} \in \{-1, 0, 1\}^n$ with $\|\mathbf{e}\|_1 < d$ is

$$N_{\mathbf{e}} = \sum_{w=0}^{d-1} \binom{n}{w} 2^w,$$

and the number of possible syndromes c is $(q-1)^d$. The vector $\mathbf{e}$ is unique if

$$\log_2 N_{\mathbf{e}} < d \log_2(q-1),$$

which is satisfied by our parameter choices. Since $\mathbf{H} = \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix}$, once the sparse $\mathbf{e}'$ is fixed, the error $\mathbf{e}''$ is uniquely determined. Since the full vector $\mathbf{e}$ is unique, the LWE secret $\mathbf{e}'$ is also unique.

This defines a well-posed LWE instance $(\mathbf{H}', \mathsf{c})$ with sparse ternary secret and error, for which the computational problem is to recover $\mathbf{e}'$ given $(\mathbf{H}', \mathsf{c})$. We assume that the number of available LWE samples is bounded by the secret dimension and take the effective number of samples to be $n-d$. The lattice estimator supports such variants of LWE. Hence, we can use it to estimate the running time of a wide variety of lattice (and combinatorial) attacks on the LLXY scheme that enable us to choose secure parameters.

Once we analysed the LLXY scheme with the lattice estimator, we observed that dual and hybrid attacks are more powerful. Therefore, we discuss how one can apply dual/hybrid attacks on our scheme.

5.1.1 Hybrid Attacks

The hybrid attack is a well-known technique for solving LWE and similar problems when the secret is sparse or small. Introduced by Howgrave-Graham [HG07] in the context of NTRU, the idea behind a hybrid attack is to reduce the dimension of a lattice problem by guessing part of the secret and solving the remaining problem using lattice reduction. This hybrid strategy typically combines brute-force guessing of a few secret coordinates with lattice-based decoding on the reduced instance, yielding better efficiency than a full-dimensional lattice attack.

In this section, we follow the row-vector convention, where the secret is a row vector and the LWE instance is given by

$$\mathbf{b} = \mathbf{s}\mathbf{A} + \mathbf{e} \pmod{q},$$

with secret $\mathbf{s} \in \mathbb{Z}_q^n$, matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, error $\mathbf{e} \in \mathbb{Z}_q^m$, and $\mathbf{b} \in \mathbb{Z}_q^m$.

Suppose we are given an LWE instance $(\mathbf{A}, \mathbf{b})$ with $\mathbf{b} = \mathbf{s}\mathbf{A} + \mathbf{e}$. A hybrid attack selects a guessing parameter k and splits the secret as a row vector $\mathbf{s} = (\mathbf{s}_1 \,\|\, \mathbf{s}_2)$, where $\mathbf{s}_1 \in \mathbb{Z}_q^{n-k}$ and $\mathbf{s}_2 \in \mathbb{Z}_q^k$. Correspondingly, we partition the matrix $\mathbf{A}$ row-wise as $\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{pmatrix}$, where $\mathbf{A}_1 \in \mathbb{Z}_q^{(n-k) \times m}$ and $\mathbf{A}_2 \in \mathbb{Z}_q^{k \times m}$.

Then we can rewrite the LWE equation as

$$\mathbf{b} = \mathbf{s}_1\mathbf{A}_1 + \mathbf{s}_2\mathbf{A}_2 + \mathbf{e} \pmod{q}.$$

Now, guess the k components of $\mathbf{s}_2$. For each guess, we obtain a reduced LWE instance

$$(\mathbf{A}_1, \tilde{\mathbf{b}} = \mathbf{b} - \mathbf{s}_2\mathbf{A}_2),$$

which depends only on $\mathbf{s}_1$ such that $\tilde{\mathbf{b}} = \mathbf{s}_1\mathbf{A}_1 + \mathbf{e}$.

The remaining task is to recover $\mathbf{s}_1$ from this reduced instance. This is typically done by embedding the problem into a lattice

$$\mathcal{L} = \{\mathbf{x} \in \mathbb{Z}^m \mid \mathbf{x} = \mathbf{s}_1\mathbf{A}_1 \pmod{q},\ \mathbf{s}_1 \in \mathbb{Z}_q^{n-k}\}.$$

Now, $\tilde{\mathbf{b}}$ is a lattice vector with a small error $\mathbf{e}$ added. Therefore, recovering $\mathbf{s}_1$ is solving a Bounded Distance Decoding (BDD) problem: find $\mathbf{x} \in \mathcal{L}$ such that $\|\tilde{\mathbf{b}} - \mathbf{x}\|_2 \leq \|\mathbf{e}\|_2$. Since norms in finite-dimensional spaces are equivalent, bounds in the ℓ_1-norm can be translated into ℓ_2-norm bounds up to a factor of at most $\sqrt{m}$.

In the hybrid attack setting, Babai's Nearest Plane Algorithm [Bab86] is used to solve the BDD instance. Nearest Plane runs in polynomial time, but requires a sufficiently good lattice basis for high success probability. This is achieved by first applying lattice reduction (e.g., BKZ) to the lattice $\mathcal{L}$ as a preprocessing step.

The main advantage of the hybrid attack is that lattice reduction is performed in dimension $n - k$ instead of n, making the BKZ step significantly more efficient. Once a sufficiently short basis is available, lattice decoding can be used to recover $\mathbf{s}_1$.

As shown by Howgrave-Graham [HG07], the guessing step can be further accelerated using meet-in-the-middle techniques. This is particularly effective when the secret has additional structure (e.g., binary or ternary secrets). Göpfert *et al.* [GvVW17] also present a generalised quantum version of the hybrid attack for LWE, where the error distribution need not be binary.

Meet-in-the-Middle Speed-up for $\mathbf{s}_2$

The guessing step can be accelerated using a Meet-in-the-Middle (MitM) strategy [HG07, Wun18], reducing the complexity from 2^k to approximately $2^{k/2}$ for binary secrets (and from 3^k to $3^{k/2}$ for ternary secrets), at the expense of additional memory.

The idea is to split the guessed part of the secret into two row vectors $\mathbf{s}_2 = (\mathbf{s}_2' \,\|\, \mathbf{s}_2'')$, with corresponding partition of $\mathbf{A}_2$ into row blocks $\mathbf{A}_2 = \begin{pmatrix} \mathbf{A}_2' \\ \mathbf{A}_2'' \end{pmatrix}$.

For each candidate $\mathbf{s}_2'$, compute the partial sum $\mathbf{u}' = \mathbf{s}_2'\mathbf{A}_2'$ and store it in a hash table indexed via a locality-sensitive hash function (LSH). The LSH partitions each coordinate into buckets of size B, ensuring that nearby vectors collide with high probability. For instance, setting $B = q/2^{v_i}$, we define

$$F : \mathbb{Z}_q \to \mathbb{Z}_{2^{v_i}}, \qquad F(u_i') = \left\lfloor \frac{u_i'}{q/2^{v_i}} \right\rfloor.$$

Next, for each candidate $\mathbf{s}_2''$, compute $\mathbf{u}'' = \mathbf{b} - \mathbf{s}_2''\mathbf{A}_2''$, and query the table for a matching bucket. A collision indicates a candidate split $\mathbf{s}_2 = (\mathbf{s}_2' \,\|\, \mathbf{s}_2'')$, which is then verified against the full equation.

In summary, the MitM strategy reduces runtime from $\mathcal{O}\big(2^k\big)$ to $\mathcal{O}\big(2^{k/2}\big)$ with $\mathcal{O}\big(2^{k/2}\big)$ memory for binary secrets (with base 2 replaced by 3 for ternary secrets). This provides a

substantial acceleration in the guessing phase. A full analysis of the hybrid attack complexity is given in Section 5.3 of Wunderer's thesis [Wun18].

We now describe how this strategy applies to the LLXY scheme.

Hybrid Attack on LLXY

Recall that in the LWE instance of the LLXY scheme (Equation (5.2)), we are given

$$\mathsf{c} = \mathbf{e}'\mathbf{H}' + \mathbf{e}'' \pmod{q-1},$$

where $\mathbf{e}' \in \{-1,0,1\}^{n-d}$, $\mathbf{e}'' \in \{-1,0,1\}^{d}$, and $\mathbf{H}' \in \mathbb{Z}_{q-1}^{(n-d)\times d}$. The goal is to recover $\mathbf{e}'$ and $\mathbf{e}''$ given c and $\mathbf{H}'$.

A hybrid attack proceeds by choosing a guessing dimension $k \in \{1,\dots,n-d-1\}$ and splitting

$$\mathbf{e}' = \begin{pmatrix}\mathbf{e}'_1 & \mathbf{e}'_2\end{pmatrix}, \quad \text{and} \quad \mathbf{H}' = \begin{pmatrix}\mathbf{H}'_1 \\ \mathbf{H}'_2\end{pmatrix},$$

where $\mathbf{e}'_1 \in \{-1,0,1\}^{n-d-k}$, $\mathbf{e}'_2 \in \{-1,0,1\}^{k}$, $\mathbf{H}'_1 \in \mathbb{Z}_{q-1}^{(n-d-k)\times d}$, and $\mathbf{H}'_2 \in \mathbb{Z}_{q-1}^{k\times d}$. This allows us to rewrite the ciphertext as

$$\mathsf{c} = \mathbf{e}'_1\mathbf{H}'_1 + \mathbf{e}'_2\mathbf{H}'_2 + \mathbf{e}'' \pmod{q-1}.$$

Subtracting the guessed part gives

$$\tilde{c} := \mathsf{c} - \mathbf{e}'_2\mathbf{H}'_2 = \mathbf{e}'_1\mathbf{H}'_1 + \mathbf{e}'' \pmod{q-1}.$$

If the guess $\mathbf{e}'_2$ is correct, then $\tilde{c}$ is close to a lattice point in the lattice $\mathcal{L}_{q-1}(\mathbf{H}'_1)$. More precisely, $\tilde{c} = \mathbf{e}'_1\mathbf{H}'_1 + (q-1)\mathbf{w} + \mathbf{e}''$ for some $\mathbf{w} \in \mathbb{Z}^d$. Since $\mathbf{e}''$ is small, $\tilde{c}$ lies close to the lattice point $\mathbf{e}'_1\mathbf{H}'_1 + (q-1)\mathbf{w}$, and a decoding algorithm can be used to recover $\mathbf{e}'_1$.

This hybrid method reduces the lattice dimension from $(n-d)$ to $(n-d-k)$ and allows for more efficient lattice reduction and decoding. The correctness of the guess for $\mathbf{e}'_2$ can be checked by verifying whether the resulting $\mathbf{e}'_1$ is ternary and whether the recomputed ciphertext matches.

The guess over $\mathbf{e}'_2$ can be further accelerated using a meet-in-the-middle technique as discussed in the previous section, where the search space is split and matched using a precomputed hash table. Table 5.2 gives the parameters obtained for the hybrid attack against the LLXY KEM using the lattice estimator [APS15]. To simulate this

meet-in-the-middle process, the lattice estimator conservatively assumes a square-root speed-up, and ignores any probability loss introduced.

LLXY Parameters			BKZ block size	Number of guessed	Guessing search space
n	d	q	(β)	coordinates (k)	
898	260	1409	228	394	$2^{150.9}$
1390	386	2209	369	646	$2^{227.5}$
1840	520	2371	612	824	$2^{361.3}$

Table 5.2: Hybrid attack parameters on the LLXY KEM according to the lattice estimator.

5.1.2 Dual Attacks

The dual attack is a standard method for solving the decisional Learning With Errors (Decision-LWE) problem by reducing it to a variant of the Short Integer Solution (SIS) problem in a *dual lattice*. Given an LWE instance $(\mathbf{A}, \mathbf{b})$ with

$$\mathbf{b} = \mathbf{sA} + \mathbf{e} \pmod{q},$$

where the secret is a row vector $\mathbf{s} \in \mathbb{Z}_q^n$, the dual attack aims to find a short vector $\mathbf{v} \in \mathbb{Z}^m$ such that

$$\mathbf{A}\mathbf{v}^\top \equiv 0 \pmod{q}.$$

This defines the scaled dual lattice

$$\mathcal{L}^* = \{\mathbf{v} \in \mathbb{Z}^m \mid \mathbf{A}\mathbf{v}^\top \equiv 0 \pmod{q}\},$$

which is (up to scaling) the lattice orthogonal to the row span of $\mathbf{A}$ in $\mathbb{Z}_q^m$.

Suppose a sufficiently short vector $\mathbf{v} \in \mathcal{L}^*$ is found using lattice reduction (e.g., BKZ with block size β). Then we compute

$$\mathbf{b}\mathbf{v}^\top = (\mathbf{sA} + \mathbf{e})\mathbf{v}^\top.$$

Expanding gives $\mathbf{b}\mathbf{v}^\top = \mathbf{s}(\mathbf{A}\mathbf{v}^\top) + \mathbf{e}\mathbf{v}^\top$. Since $\mathbf{A}\mathbf{v}^\top \equiv 0 \pmod{q}$, we obtain $\mathbf{b}\mathbf{v}^\top \equiv \mathbf{e}\mathbf{v}^\top \pmod{q}$.

Since both $\mathbf{v}$ and $\mathbf{e}$ are short, the value $\mathbf{v}\mathbf{b}^\top$ is statistically far from uniform modulo q, allowing one to distinguish LWE samples from uniform random samples. Hence, the dual attack yields a distinguisher for LWE. Rather than relying on the classical search-to-decision

reduction [Reg09], modern lattice cryptanalysis embeds this distinguisher into a structural framework that simplifies the target instance. Specifically, techniques such as sparsification or dimension reduction map the initial BDD instance that underlies the LWE instance to a new BDD instance defined over a lower-dimensional or sparser lattice. This transformed, easier instance is then solved to recover the secret. For in-depth structural details of this method, see [DP23, KKN$^+$26].

Albrecht [Alb17] introduced the *dual hybrid attack*, which improves the basic dual attack by combining lattice reduction with partial guessing of the secret. The key idea is that guessing part of the secret reduces the effective LWE dimension, making the lattice reduction step cheaper.

Concretely, split the secret as a row vector $\mathbf{s} = (\mathbf{s}_1 \,\|\, \mathbf{s}_2)$, with $\mathbf{s}_1 \in \mathbb{Z}_q^{n-k}$ and $\mathbf{s}_2 \in \mathbb{Z}_q^k$. Accordingly, partition the matrix row-wise as $\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{pmatrix}$, where $\mathbf{A}_1 \in \mathbb{Z}_q^{(n-k)\times m}$ and $\mathbf{A}_2 \in \mathbb{Z}_q^{k\times m}$. Then the LWE equation becomes

$$\mathbf{b} = \mathbf{s}_1\mathbf{A}_1 + \mathbf{s}_2\mathbf{A}_2 + \mathbf{e} \pmod q.$$

The attacker guesses $\mathbf{s}_2$, obtaining a reduced instance

$$\mathbf{b}' = \mathbf{b} - \mathbf{s}_2\mathbf{A}_2 = \mathbf{s}_1\mathbf{A}_1 + \mathbf{e} \pmod q.$$

If the guess is correct, this yields an LWE instance of dimension $n - k$. This reduction is the key idea of the dual hybrid attack: : it trades the cost of guessing k secret coordinates for a smaller lattice dimension.

Once the dimension is reduced, the attacker applies the dual attack to the subsystem defined by $(\mathbf{A}_1, \mathbf{b}')$, where $\mathbf{b}' = \mathbf{b} - \mathbf{s}_2\mathbf{A}_2$. In particular, the scaled dual lattice is defined as

$$\mathcal{L}_{q,c}(\mathbf{A}_1) = \{(\mathbf{v}, \mathbf{y}) \in \mathbb{Z}^m \times (1/c \cdot \mathbb{Z})^{n-k} \mid \mathbf{A}_1\mathbf{v}^\top \equiv c\mathbf{y} \pmod q\},$$

for some scaling parameter c. Lattice reduction is used to find short vectors $(\mathbf{v}, \mathbf{y}) \in \mathcal{L}_{q,c}(\mathbf{A}_1)$. Applying such a vector to the reduced LWE instance yields

$$\mathbf{b}'\mathbf{v}^\top = (\mathbf{s}_1\mathbf{A}_1 + \mathbf{e})\mathbf{v}^\top = \mathbf{s}_1(\mathbf{A}_1\mathbf{v}^\top) + \mathbf{e}\mathbf{v}^\top.$$

Using the defined scaled dual lattice, we obtain $\mathbf{A}_1\mathbf{v}^\top \equiv c\mathbf{y} \pmod q$, and therefore

$$\mathbf{b}'\mathbf{v}^\top \equiv c\mathbf{s}_1\mathbf{y} + \mathbf{e}\mathbf{v}^\top \pmod q.$$

When $(\mathbf{v}, \mathbf{y})$ are sufficiently short, the term $c\mathbf{s}_1\mathbf{y} + \mathbf{e}\mathbf{v}^\top$ remains small. Thus, for the correct guess of $\mathbf{s}_2$, the value $\mathbf{b}'\mathbf{v}^\top$ is distinguishable from uniform distribution over $\mathbb{Z}_q$. In this way, the dual hybrid attack applies the dual attack in reduced dimension $n-k$, achieving a significantly improved complexity.

As with other hybrid approaches, the guessing phase can be accelerated using meet-in-the-middle (MitM) strategies that trade memory for runtime. Cheon *et al.* [CHHS19] adapt the MitM strategy of Howgrave-Graham [HG07] to the dual hybrid setting using locality-sensitive hashing. The idea is to split the guessed coordinates further, writing $\mathbf{s}_2 = (\mathbf{s}_2' \parallel \mathbf{s}_2'')$ with each half of the size $k/2$. For each candidate $\mathbf{s}_2'$, the attacker computes and stores the intermediate value $\mathbf{s}_2'\mathbf{A}_2'$ in a hash table indexed via a locality-sensitive hash function that preserves approximate equality modulo q. Then for each candidate $\mathbf{s}_2''$, compute $\mathbf{b} - \mathbf{s}_2''\mathbf{A}_2''$, and check for a matching entry in the table up to the tolerated error.A collision corresponds to a consistent guess for $(\mathbf{s}_2', \mathbf{s}_2'')$, reconstructing $\mathbf{s}_2$. This MitM approach reduces the time complexity of guessing $\mathbf{s}_2$ from $\mathcal{O}\big(q^k\big)$ to $\mathcal{O}\big(q^{k/2}\big)$, at the cost of $\mathcal{O}\big(q^{k/2}\big)$ memory, yielding a favourable time–memory trade-off.

Dual Hybrid Attack on LLXY

We now apply the dual hybrid attack to the LLXY scheme, where the LWE instance is given by the matrix-vector pair $(\mathbf{H}', \mathsf{c})$ with

$$\mathsf{c} = \mathbf{e}'\mathbf{H}' + \mathbf{e}'' \pmod{q-1},$$

and $\mathbf{e}' \in \{-1, 0, 1\}^{n-d}$, $\mathbf{e}'' \in \{-1, 0, 1\}^d$, and $\mathbf{H}' \in \mathbb{Z}_{q-1}^{(n-d)\times d}$.

To apply the dual hybrid attack, select a guessing parameter k, and split $\mathbf{e}' = \begin{pmatrix}\mathbf{e}_1' & \mathbf{e}_2'\end{pmatrix}$ with $\mathbf{e}_2' \in \{-1, 0, 1\}^k$, and likewise partition $\mathbf{H}'$ as

$$\mathbf{H}' = \begin{pmatrix}\mathbf{H}_1' \\ \mathbf{H}_2'\end{pmatrix}, \quad \text{where } \mathbf{H}_1' \in \mathbb{Z}_{q-1}^{(n-d-k)\times d},\ \mathbf{H}_2' \in \mathbb{Z}_{q-1}^{k\times d}.$$

The new formulation of the ciphertext is

$$\mathsf{c} = \mathbf{e}_1'\mathbf{H}_1' + \mathbf{e}_2'\mathbf{H}_2' + \mathbf{e}''. \tag{5.3}$$

Construct the scaled dual lattice,

$$\mathcal{L}_{q-1,c}(\mathbf{H}_1') = \left\{(\mathbf{v}, \mathbf{u}) \in \mathbb{Z}^d \times \left(\tfrac{1}{c}\mathbb{Z}\right)^{n-d-k} \;\middle|\; \mathbf{H}_1'\mathbf{v}^\top \equiv c\mathbf{u} \pmod{q-1}\right\},$$

where $\mathbf{v}^\top$ is the transpose of $\mathbf{v}$. The goal is to find a short vector $(\mathbf{y}_1, \mathbf{y}_2) \in \mathcal{L}_{q-1,c}(\mathbf{H}'_1)$ using lattice reduction, typically modelled as BKZ with a given root Hermite factor δ. The length of the vector determines whether the product with the ciphertext remains small. Specifically, using Equation (5.3) compute

$$\mathsf{c} \cdot \mathbf{y}_1^\top = \mathbf{e}'_1 \mathbf{H}'_1 \mathbf{y}_1^\top + \mathbf{e}'_2 \mathbf{H}'_2 \mathbf{y}_1^\top + \mathbf{e}'' \mathbf{y}_1^\top.$$

Using the lattice relation $\mathbf{H}'_1 \mathbf{y}_1^\top \equiv c\mathbf{y}_2 \pmod{q-1}$, this becomes

$$\mathsf{c} \cdot \mathbf{y}_1^\top \equiv c\mathbf{e}'_1 \mathbf{y}_2^\top + \mathbf{e}'_2 \mathbf{H}'_2 \mathbf{y}_1^\top + \mathbf{e}'' \mathbf{y}_1^\top \pmod{q-1}.$$

Thus, for each sufficiently short vector $(\mathbf{y}_1, \mathbf{y}_2)$, the term $c\mathbf{e}'_1 \mathbf{y}_2^\top + \mathbf{e}'' \mathbf{y}_1^\top$ remains small for LWE samples. For a correct guess of $\mathbf{e}'_2 \in \{-1, 0, 1\}^k$, one can distinguish between LWE samples and uniform distribution over $\mathbb{Z}_{q-1}$.

One can see that the dual-hybrid format of the LWE instance is easier than the original because it reduces the effective lattice dimension by guessing a small subset of secret components. By splitting $\mathbf{e}'$ into $\mathbf{e}'_1$ and $\mathbf{e}'_2$ and guessing $\mathbf{e}'_2$, lattice reduction is applied only to the smaller lattice corresponding to $\mathbf{e}'_1$. This splits the attack cost between guessing and lattice reduction. Lattice reduction cost is estimated using BKZ on the reduced-dimension lattice defined by $\mathbf{H}'_1$ and is exponential in the BKZ block size β. The guessing phase is realised using a meet-in-the-middle process and assume a square-root speed-up in the cost (3^k for exhaustive search and $3^{k/2}$ for MitM strategy). Table 5.3 summarises the cost for lattice reduction and the cost for guessing $\mathbf{e}'_2$ in the dual hybrid attack for the LLXY KEM parameters we chose according to the lattice estimator [APS15].

LLXY Parameters			Cost for lattice reduction	Cost of guessing
n	d	q		
898	260	1409	$2^{139.9}$	$2^{137.9}$
1390	386	2209	$2^{207.5}$	$2^{208.6}$
1840	520	2371	$2^{275.0}$	$2^{269.0}$

Table 5.3: Dual Hybrid attack costs from the lattice estimator on the LLXY KEM.

5.1.3 Lattice Estimator Results

Recall from Section 5.1, the LWE instance for the LLXY is defined as Equation (5.2),

$$\mathsf{c} = \mathbf{e}'\mathbf{H}' + \mathbf{e}''.$$

A key feature of the LLXY setting is that operations are carried out in $\mathbb{Z}_{q-1}$ rather than $\mathbb{Z}_q$ for a prime q. For the analysis, we consider sparse ternary secrets ($\mathbf{e}'$), and error vectors ($\mathbf{e}''$). We assume that the number of $+1$'s and -1's in the secret and error vectors are equal. In particular, the number of $+1$'s (respectively -1's) in the error vector $\mathbf{e}'' \in \{-1, 0, 1\}^d$ is assumed to be

$$\frac{(d-1)d}{2n},$$

and the number of $+1$'s (respectively -1's) in the secret vector $\mathbf{e}' \in \{-1, 0, 1\}^{n-d}$ is assumed to be

$$\frac{(d-1)(n-d)}{2n}.$$

The lattice attacks are analysed using the Lattice Estimator [APS15], which makes the following assumptions for dual and dual-hybrid attacks:

- **Reduced basis shape:** Geometric Series Assumption (GSA) is used to model the shape of the BKZ-reduced lattice basis.
- **Meet-in-the-middle (MitM) speed-up:** A square-root speed up is assumed when enumerating guesses for parts of the secret.
- **MitM success probability:** Assumed to be 1 for correct guesses.
- **Memory cost:** Ignored in the cost estimate.

Table 5.4 gives the estimated cost of the above lattice attacks for our chosen set of parameters. For comparison, [LLXY20] used only the Unique-SVP lattice attack when selecting parameters. We include the Unique-SVP cost in the table to illustrate that hybrid and dual-hybrid attacks are more effective in this setting.

Lattice attack	(n, d, q)		
	$(898, 260, 1409)$	$(1390, 386, 2209)$	$(1840, 520, 2371)$
Unique-SVP	$2^{151.3}$	$2^{226.4}$	$2^{299.7}$
Dual hybrid	$2^{140.2}$	$2^{209.2}$	$2^{275.0}$
Hybrid (MitM)	$2^{140.6}$	$2^{199.6}$	$2^{262.1}$

Table 5.4: Lattice estimator costs for proposed parameters.

5.2 Decoding attacks

Recall that the LLXY scheme is of the form $\mathsf{c} = \mathbf{eH} \pmod{q-1}$ where $\mathbf{H}$ is the parity check matrix. Solving for $\mathbf{e}$ is a Syndrome Decoding Problem (SDP). The best algorithmic paradigm that we know today for solving the SDP is a class of algorithms called Information Set Decoding (ISD). The underlying idea of the ISD algorithms is to solve a dimension-reduced standard form of a decoding problem instance instead of its original form. The first ISD algorithm is due to Prange [Pra62] which was later improved by Stern and Dumer [Ste88, Dum91]. Despite many efforts on designing practical decoding algorithms, the state-of-the-art algorithms in the literature [BM18, MO15, BJMM12] are exponential in the number of errors that have to be corrected and all can be viewed as a refinement of the original Prange's algorithm.

It is worth noting that the state-of-the-art in information set decoding is mainly developed for codes over binary fields, and is not necessarily expected to extend to codes over larger fields and different metrics than Hamming distance. From our investigations we believe the most promising approach for our problem is the nearest neighbour approach propose by May and Ozerov [MO15] and later optimised in [BM17b, BM18]. May-Ozerov propose to decode random binary linear codes with nearest neighbour search and obtain better running time than the Stern's algorithm for information set decoding. In particular, they solve the following problem [MO15].

Definition 5.1 (NN problem)**.** *Let $n \in \mathbb{N}, 0 < \gamma < 1/2$ and $0 < \lambda < 1$. In the (n, γ, λ)-Nearest Neighbour (NN) problem, we are given γ and two lists $L, R \subset \mathbb{F}_2^n$ of equal size $2^{\lambda n}$ with uniform and pairwise independent vectors. If there exists a pair $(u^*, v^*) \in L \times R$ with Hamming distance $n\gamma$, the task is to output a list C that contains (u^*, v^*).*

Lapiha proposed applying decoding algorithms to LLXY setting and studied syndrome decoding in the ℓ_1 metric, which in this setting is a case of the Inhomogeneous Short Integer Solution (ISIS) problem from lattice cryptography. In Section 4.3 of [Lap21] Lapiha essentially re-discovers an algorithm for ISIS developed by Bai et al. [BGLS19], which in turn is closely related to previous work by May and Ozerov that uses nearest neighbour search. In the ISIS case, the nearest neighbour search modulo an integer is done by matching on high order bits (the same method is used in both [BGLS19, Lap21]). We believe this method gives a practical and efficient solution to the problem in the LLXY setting as well and therefore, in this work we adopt similar approaches to provide an updated security analysis for the LLXY scheme.

We make two observations about applying the nearest neighbour algorithm to the LLXY system. First, the May-Ozerov algorithm focuses on linear codes over $\mathbb{F}_2$ whereas the LLXY

system works on larger $(q-1)$-ary codes. Secondly, LLXY is based on the ℓ_1 norm of the error vectors, while the majority of decoding algorithms in the literature operate with Hamming weight. Therefore, one needs to modify the Near Neighbour algorithm to fit with this setting. Once these modifications are made (as was done in [BGLS19, Lap21]) one sees that the nearest neighbour approach works very well for our problem compared with the previous case of codes over binary fields under Hamming distance.

In the LLXY KEM, we are given a parity check matrix $\mathbf{H} \in \mathbb{Z}_{q-1}^{n \times d}$ in standard form and a syndrome $\mathsf{c} \in \mathbb{Z}_{q-1}^{d}$, the task is to determine the error vector $\mathbf{e} \in \{-1, 0, 1\}^n$ that satisfies the equation

$$\mathbf{e}\mathbf{H} = \mathbf{e} \cdot \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix} = \mathsf{c} \text{ where } \|\mathbf{e}\|_1 = w \leq d - 1. \tag{5.4}$$

Stern's algorithm searches for $\mathbf{e}$ where $\|\mathbf{e}\|_1 = p > 0$ on the first $n - d$ coordinates and $w - p$ on the last d coordinates. Since this exact splitting of weights does not always happen, it is necessary to apply a random permutation π to $\mathbf{e}$ (by permuting the rows of $\mathbf{H}$ and then re-converting to standard form). With a certain probability the weights split as desired and the algorithm finds them.

We present and analyse two versions of the nearest neighbour decoding algorithm. In particular, we consider a basic two-list variant and an extended four-list variant inspired by the approach in [BGLS19] and give rough lower bound estimates of complexity. This comparison highlights how the performance of the algorithm improves when the number of lists is increased, thereby offering insights into the trade-off between complexity and success probability of the decoding attack.

5.2.1 Two-List Version

The simplest instantiation of the Near Neighbour approach is the two-list version. The error vector is partitioned as $\mathbf{e} = \begin{pmatrix} \mathbf{e}' & \mathbf{e}'' \end{pmatrix}$, and Equation (5.4) can be rewritten in the form

$$\mathsf{c} = \begin{pmatrix} \mathbf{e}' & \mathbf{e}'' \end{pmatrix} \begin{pmatrix} \mathbf{H}' \\ \mathbf{I}_d \end{pmatrix} = \mathbf{e}'\mathbf{H}' + \mathbf{e}''.$$

We further decompose $\mathbf{e}'$ as $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ with $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)} \in \{-1, 0, 1\}^{n-d}$. Substituting into the above yields

$$(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)})\mathbf{H}' + \mathbf{e}'' = \mathsf{c}. \tag{5.5}$$

The splitting of $\mathbf{e}'$ is done positionally: the vector $\mathbf{e}_1^{(1)}$ is non-zero only on the first $(n-d)/2$ coordinates, while $\mathbf{e}_2^{(1)}$ is non-zero only on the remaining coordinates, i.e., positions $\{(n-d)/2+1,\ldots,n-d\}$. The vector $\mathbf{e}''$ remains as an element of $\{-1,0,1\}^d$.

We choose $p \approx \frac{(d-1)(n-d)}{n}$ to be an even integer, representing an estimate of the number of non-zero entries in the first $n-d$ coordinates of the error vector $\mathbf{e}$. The attack proceeds under the assumption that

$$\|\mathbf{e}_1^{(1)}\|_1 = \frac{p}{2}, \quad \|\mathbf{e}_2^{(1)}\|_1 = \frac{p}{2},$$

so that $\|\mathbf{e}'\|_1 = p$ and consequently $\|\mathbf{e}''\|_1 = w - p$. We refer to this choice of partitioning the weight of $\mathbf{e}$ as the *desired weight distribution*. As with Stern's algorithm, a random permutation is applied on the parity check matrix and the algorithm is repeated until it succeeds.

Unlike the Dumer or BJMM methods, this approach uses vectors $\mathbf{e}_1^{(1)}$ and $\mathbf{e}_2^{(1)}$ of length $n-d$ (as opposed to $n-d+l$), which reduces the size of the search space. Moreover, since $p < w$, the required ℓ_1-norm on the first $n-d$ coordinates is relatively small. As a result, fewer iterations are needed to find a permutation π that satisfies the desired weight distribution. On the other hand, this advantage comes at a cost: the approximate matching required in this method is generally more computationally expensive than the exact matching used in traditional ISD algorithms.

We write,

$$\mathbf{e}_1^{(1)}\mathbf{H}' + \mathbf{e}_2^{(1)}\mathbf{H}' + \mathbf{e}'' = \mathsf{c} \quad \Longrightarrow \quad \mathbf{e}_1^{(1)}\mathbf{H}' \approx \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}'. \tag{5.6}$$

Here, $\mathbf{e}'' \in \{-1,0,1\}^d$ and the set of possible values of $\mathbf{e}''$ is

$$\mathcal{E} = \{\mathbf{x} \in \{-1,0,1\}^d \mid \|\mathbf{x}\|_1 = w - p\}.$$

Thus, finding an approximate merge $\mathbf{e}_1^{(1)}\mathbf{H}' \approx \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}'$ is equivalent to checking whether

$$\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' - \mathbf{e}_1^{(1)}\mathbf{H}' \in \mathcal{E}.$$

The search is carried out using two lists:

$$L_1^{(1)} = \left\{ \left(\mathbf{e}_1^{(1)}\mathbf{H}', \mathbf{e}_1^{(1)}\right) \;\middle|\; \begin{array}{l} \mathbf{e}_1^{(1)}\mathbf{H}' \in \mathbb{Z}_{q-1}^d, \quad \mathbf{e}_1^{(1)} \in \{-1,0,1\}^{\frac{n-d}{2}} \times \{\mathbf{0}\}^{\frac{n-d}{2}}, \\ \|\mathbf{e}_1^{(1)}\|_1 = p/2 \end{array} \right\},$$

$$L_2^{(1)} = \left\{ \left(\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}', \mathbf{e}_2^{(1)}\right) \;\middle|\; \begin{array}{l} \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' \in \mathbb{Z}_{q-1}^d, \quad \mathbf{e}_2^{(1)} \in \{\mathbf{0}\}^{\frac{n-d}{2}} \times \{-1,0,1\}^{\frac{n-d}{2}}, \\ \|\mathbf{e}_2^{(1)}\|_1 = p/2 \end{array} \right\}.$$

The attack proceeds by sorting $L_1^{(1)}$ with respect to the first coordinate and, for each element $\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' \in L_2^{(1)}$, searching for an approximate match $\mathbf{e}_1^{(1)}\mathbf{H}' \in L_1^{(1)}$. If a close match is found, we verify that $\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)} \in \{-1, 0, 1\}^{n-d}$ and $\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' - \mathbf{e}_1^{(1)}\mathbf{H}' \in \mathcal{E}$. If both conditions hold, we set $\mathbf{e}'' = \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' - \mathbf{e}_1^{(1)}\mathbf{H}'$ and add the pair $(\mathbf{e}'', \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)})$ to a new list $L_1^{(2)}$. Each element of $L_1^{(2)}$ thus corresponds to a valid full error vector $\mathbf{e} = \begin{pmatrix} \mathbf{e}' & \mathbf{e}'' \end{pmatrix}$, where $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$.

$$L_1^{(2)} = \left\{ \left(\mathbf{e}'', \mathbf{e}'\right) \;\middle|\; \begin{array}{l} \mathbf{e}'' = \mathsf{c} - \mathbf{e}'\mathbf{H}' \in \{-1, 0, 1\}^d, \quad \mathbf{e}' \in \{-1, 0, 1\}^{n-d}, \\ \|\mathbf{e}'\|_1 = p, \;\; \|\mathsf{c} - \mathbf{e}'\mathbf{H}'\|_1 = w - p \end{array} \right\}.$$

Algorithm 4 Two-list Near Neighbour Algorithm

Input: $L_1^{(1)}, L_2^{(1)}$ as defined above.
Output: $L_1^{(2)}$

1: Initialise $L_1^{(2)} = \emptyset$.
2: Sort $L_1^{(1)}$ with respect to first coordinate $\mathbf{e}_1^{(1)}\mathbf{H}'$.
3: **for** $(\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}', \mathbf{e}_2^{(1)}) \in L_2^{(1)}$ **do**
4: Find $(\mathbf{e}_1^{(1)}\mathbf{H}', \mathbf{e}_1^{(1)}) \in L_1^{(1)}$ such that $\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' \approx \mathbf{e}_1^{(1)}\mathbf{H}'$.
5: **if** $\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)} \in \{-1, 0, 1\}^{n-d}$ and $\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' - \mathbf{e}_1^{(1)}\mathbf{H}' \in \mathcal{E}$ **then**
6: Compute $\mathbf{e}'' = \mathsf{c} - (\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)})\mathbf{H}'$.
7: Add $(\mathbf{e}'', \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)})$ to $L_1^{(2)}$.
8: **end if**
9: **end for**
10: **return** $L_1^{(2)}$

Complexity Analysis

In this section we give lower bounds for the running time estimate of the two-list decoding attack. To analyse the complexity of the algorithm, first observe that the construction of lists $L_1^{(1)}$ and $L_2^{(1)}$ and the existence of $\mathbf{e}''$ with the correct weight distribution, relies entirely on the probabilistic assumption that the vector $\mathbf{e}$ originally had the desired distribution. The probability of obtaining the desired weight distribution (i.e., a "good splitting") is given by

$$\Pr(\text{good splitting}) = \frac{\binom{(n-d)/2}{p/2}^2 \binom{d}{w-p}}{\binom{n}{d-1}}. \tag{5.7}$$

The inverse of this probability gives the expected number of repetitions required to obtain a vector $\mathbf{e}$ with the desired weight distribution. In each repetition, we generate the

two uniform and pairwise independent lists $L_1^{(1)}$ and $L_2^{(1)}$, each of size

$$|L_1^{(1)}| = |L_2^{(1)}| = 2^{p/2}\binom{(n-d)/2}{p/2}, \tag{5.8}$$

and the time to construct each list is proportional to its size.

Lemma 5.1. *Let notation be as above. Then the two-list NN algorithm (Algorithm 4) has a lower bound complexity estimate of*

$$\mathcal{O}\left(|L_1^{(1)}|\log(|L_1^{(1)}|) + |L_2^{(1)}||\mathcal{E}|\log(|L_1^{(1)}|) + |L_1^{(1)}|\cdot|L_2^{(1)}|\frac{|\mathcal{E}|}{(q-1)^d}\right). \tag{5.9}$$

Proof. Line 2 of the algorithm requires sorting $L_1^{(1)}$ with respect to the first coordinate $\mathbf{e}_1^{(1)}\mathbf{H}'$, which can be done in $\mathcal{O}\left(|L_1^{(1)}|\log(|L_1^{(1)}|)\right)$ operations.

Step 4 of the algorithm is the NN search. The Near Neighbour search is the detection of values $\mathbf{e}_1^{(1)}\mathbf{H}'$ in the sorted list such that $\mathbf{e}_1^{(1)}\mathbf{H}' \approx \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}'$. This can be implemented in several ways, e.g. by trying all $\mathbf{e}'' \in \mathcal{E}$ or by hashing using most significant bits (see Section 4.2 of [BGLS19]). The running time depends on this choice. If one uses the method of trying all the possible values of $\mathbf{e}'' \in \mathcal{E}$, and for each $(\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}') \in L_2^{(1)}$, search for matching candidates from $L_1^{(1)}$, the cost of Step 4 in Algorithm 4 is

$$|L_2^{(1)}|\cdot|\mathcal{E}|\log(|L_1^{(1)}|) \qquad \text{where} \quad |\mathcal{E}| = 2^{w-p}\binom{d}{w-p}.$$

Next, consider the total number of matches detected between $L_1^{(1)}$ and $L_2^{(1)}$. Heuristically, for each pair in $L_2^{(1)}$, we expect $|L_1^{(1)}|/(q-1)^d$ matches in $L_1^{(1)}$ and for each match we have to check if the error is of the correct weight. The correctness check in line 5 of the Algorithm 4 is performed

$$|L_2^{(1)}|\cdot\left\lceil\frac{|L_1^{(1)}|\cdot|\mathcal{E}|}{(q-1)^d}\right\rceil \tag{5.10}$$

times.

Step 6 and 7 only execute when the checks in Step 5 pass. For simplicity, and since we are considering the lower bounds, we bound the running time proportional to above steps. Therefore, the Near Neighbour algorithm (Algorithm 4) performs

$$\mathcal{O}\left(|L_1^{(1)}|\log(|L_1^{(1)}|) + |L_2^{(1)}||\mathcal{E}|\log(|L_1^{(1)}|) + |L_1^{(1)}|\cdot|L_2^{(1)}|\frac{|\mathcal{E}|}{(q-1)^d}\right)$$

arithmetic operations. □

Hash function. We observed that trying all $\mathbf{e}'' \in \mathcal{E}$ is computationally intensive. To acquire a speed-up in our computations we consider hashing using the most significant bits. We define the hash function

$$F : \mathbb{Z}_{q-1} \to \mathbb{Z}_{2^\nu} \qquad \text{by} \quad F(x) = \left\lfloor \frac{x}{(q-1)/2^\nu} \right\rfloor,$$

where ν is a parameter indicating the number of most significant bits used. We adopt the terminology of [BGLS19], adapted to ternary error vectors. An integer $a \in \mathbb{Z}$ is called *borderline* if $F(a) \neq F(a+e)$ or $F(a) \neq F(a-e)$ for a possible error value $e \in \{-1, 0, 1\}$. We can then extent F to vectors in $\mathbb{Z}_{q-1}^d$. Bai et al. [BGLS19] define the set of all patterns of the most significant bits that would arise by adding valid errors to the corresponding coordinates of a vector $\mathbf{v}$ by

$$\text{Flips}(\mathbf{v}) = \{F(\mathbf{v} \pm \mathbf{e}) \ : \ \mathbf{e} \in \mathcal{E}\}.$$

One does not have to loop through all $\mathbf{e} \in \mathcal{E}$ to compute $\text{Flips}(\mathbf{v})$ and it is enough to check entries of $\mathbf{v}$ that are on the borderline. The expected size of the set can be calculated by counting the number of borderline integers in $\mathbb{Z}_{q-1}$ and checking whether they going to "flip" with the error. When $\mathbf{e} \in \{-1, 0, 1\}$, there are $2^{\nu+1}$ borderline integers in $\mathbb{Z}_{q-1}$. Therefore, the expected size of the set $\text{Flips}(\mathbf{v})$ for vectors $\mathbf{v} \in \mathbb{Z}_{q-1}^d$ is $\left(1 + \frac{2^{\nu+1}}{q-1}\right)^d$.

The Near Neighbour algorithm searches for matches of hash values between two lists. More precisely, first compute $F(\mathbf{v})$ for every $\mathbf{v} = \mathbf{e}_1^{(1)}\mathbf{H}'$ in $L_1^{(1)}$ and sort these values. Then for each $\mathbf{v} = \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}'$ in $L_2^{(1)}$ compute $\text{Flips}(\mathbf{v})$, and for each $\mathbf{u} \in \text{Flips}(\mathbf{v})$, search for occurrences of $\mathbf{u}$ in the sorted list. Finally, for each match, check whether, $\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' - \mathbf{e}_1^{(1)}\mathbf{H}' \in \mathcal{E}$.

With this hashing strategy, the running time estimate, Equation (5.9) can be rewrite as

$$\mathcal{O}\left(|L_1^{(1)}| \log(|L_1^{(1)}|) + |L_2^{(1)}||\text{Flips}(\mathbf{v})| \log(|L_1^{(1)}|) + |L_1^{(1)}| \cdot |L_2^{(1)}| \frac{|\text{Flips}(\mathbf{v})|}{2^{d\nu}} \right). \tag{5.11}$$

Since the aim is to get a lower bound, we treat few operations as constant time. The first term of Equation (5.11) corresponds to sorting $L_1^{(1)}$, assuming that computing $F(\mathbf{e}_1^{(1)}\mathbf{H}')$ is constant time. Next for each pair $(\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}', \mathbf{e}_2^{(1)}) \in L_2^{(1)}$, we have to compute $\text{Flips}(\mathbf{v})$ where $\mathbf{v} = \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}'$ and needs to look up the value in the sorted list. This gives the second component of the equation. For each hash match, there are on average, $|L_1^{(1)}|/2^{d\nu}$ different candidates from the first list to check. Hence, the expected number of values that we need to

handle is around $|L_1^{(1)}| \cdot |L_2^{(1)}| \frac{|\text{Flips}(\mathbf{v})|}{2^{d\nu}}$, followed by the check that $\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' - \mathbf{e}_1^{(1)}\mathbf{H}' \in \mathcal{E}$. We consider the number of bit-operations required for the final correctness check to be constant. A complete running time analysis of the NN algorithm is given in Lemma 3 of [BGLS19].

We note that, the average cost of the ISD attack is determined by two components:

- the cost of constructing the lists $L_1^{(1)}, L_2^{(1)}$ – Equation (5.8),
- the cost of the Near Neighbour algorithm – Equation (5.11).

Let,

$$T = \max\{\text{Cost of list construction}, \text{Cost of NN algorithm}\}.$$

Since the algorithm must be repeated until a "good splitting" occurs with probability $\Pr(\text{good splitting})$ (Equation (5.7)), the expected (average) cost of the ISD attack is

$$\text{Average ISD cost} \;=\; \frac{T}{\Pr(\text{good splitting})}. \tag{5.12}$$

This completes the description of the two-list Near Neighbour ISD attack. We now turn to a four-lists decoding, in order to explore the improvements achievable by increasing the number of lists.

5.2.2 Four-List Version

The state-of-the-art ISD algorithms use more than two lists to achieve smaller list sizes and hence better running time than the two-list version. For example, Both and May [BM18] propose a depth-2 decoding algorithm (which we refer to as the four-list version for consistency), and later generalise the idea to arbitrary depths. Their method speeds up the May-Ozerov [MO15] and BJMM [BJMM12] algorithms by replacing exact matching with approximate matching at all stages, thereby achieving improved asymptotic performance.

In this thesis, we follow the work in [BGLS19] in which they offer a way of subdividing the problem while oversampling the errors to increase the number of potential solutions. We restrict our focus to the four-list version and leave the study of deeper variants to future work.

The four-list version follows the same initial splitting of the error vector as in the two-list version. Specifically, when $\mathbf{H}$ is in the standard form, the error vector $\mathbf{e} \in \{-, 1, 0, 1\}^n$ where $\|\mathbf{e}\|_1 = w \leq d - 1$ is partitioned as

$$\mathbf{e} = \begin{pmatrix} \mathbf{e}' & \mathbf{e}'' \end{pmatrix} \quad \text{where} \quad \mathbf{e}' \in \{-, 1, 0, 1\}^{n-d} \text{ and } \mathbf{e}'' \in \{-, 1, 0, 1\}^d.$$

Then the vector $\mathbf{e}'$ is further decomposed as $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ with $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)} \in \{-1, 0, 1\}^{n-d}$. This splitting is different to the two-list version, because now we split by weight, or more specifically in our context, according to the ℓ_1-norm. Substituting this into the decoding equation yields:

$$(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)})\mathbf{H}' + \mathbf{e}'' = \mathsf{c}. \tag{5.13}$$

The key idea of the attack is to use four lists with two levels of recursion to find a solution $\mathbf{e}' \in \{-, 1, 0, 1\}^{n-d}$ with ℓ_1-norm p where we choose $p = \frac{(d-1)(n-d)}{n}$. The first level of recursion splits $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$, and we enforce $\|\mathbf{e}_1^{(1)}\|_1 = \|\mathbf{e}_2^{(1)}\|_1 = p_1$ where $p_1 = p/2 + \alpha$ for a suitably chosen α. The second level of recursion splits $\mathbf{e}_1^{(1)} = \mathbf{e}_1^{(0)} + \mathbf{e}_2^{(0)}$ by positions, where $\mathbf{e}_1^{(0)}$ is non-zero only on the first $(n-d)/2$ coordinates and $\mathbf{e}_2^{(0)}$ is non-zero only on the $\{(n-d)/2 + 1, \ldots, (n-d)\}$ coordinates with $\|\mathbf{e}_1^{(0)}\|_1 = \|\mathbf{e}_2^{(0)}\|_1 = p_1/2$. Similarly, split $\mathbf{e}_2^{(1)}$ into $\mathbf{e}_3^{(0)}$ and $\mathbf{e}_4^{(0)}$.

We also introduce an additional split by weight within the d coordinates of the error vector $\mathbf{e}''$ which has an ℓ_1-norm $w - p$. Specifically, we partition the d coordinates as l_1 and $l_2 := d - l_1$, and correspondingly partition $\mathbf{e}'' = \begin{pmatrix}\mathbf{e}_1'' & \mathbf{e}_2''\end{pmatrix}$, where $\mathbf{e}_1'' \in \{-1, 0, 1\}^{l_1}$ and $\mathbf{e}_2'' \in \{-1, 0, 1\}^{l_2}$. Suppose $\|\mathbf{e}_1''\|_1 = w_1 = \frac{(d-1)l_1}{n}$. We split $\mathbf{e}_1'' = \mathbf{e}_{11}'' + \mathbf{e}_{12}''$ such that $\|\mathbf{e}_{11}''\|_1 = \|\mathbf{e}_{12}''\|_1 = w_1'' = w_1/2 + \beta$. Note that l_1 and β are parameters to be optimised. This allows $\mathbf{e}_2''$ to have an ℓ_1-norm of $w_2 := w - p - w_1$.

Outline of the Algorithm

The goal is to construct the final list

$$L_1^{(2)} = \left\{ (\mathbf{e}', \mathbf{e}'') \;\middle|\; \begin{array}{l} \mathbf{e}' \in \{-1, 0, 1\}^{n-d},\ \mathbf{e}'' = \mathsf{c} - \mathbf{e}'\mathbf{H}' \in \{-1, 0, 1\}^{d}, \\ \|\mathbf{e}'\|_1 = p,\ \|\mathbf{e}''\|_1 = w - p \end{array} \right\},$$

whose elements satisfy the required weight conditions. The construction of $L_1^{(2)}$ proceeds in two levels:

Level 0. Start with four lists $L_i^{(0)}$ $(i = 1, \ldots, 4)$. To illustrate, we describe the construction of the first two and the others are analogous. For $L_1^{(0)}$, enumerate all vectors $\mathbf{e}_1^{(0)}$ that are non-zero only on the first $(n-d)/2$ coordinates, with $\|\mathbf{e}_1^{(0)}\|_1 = p_1/2$. For each such vector, compute $\mathbf{e}_1^{(0)}\mathbf{H}'$.

$$L_1^{(0)} = \left\{ (\mathbf{e}_1^{(0)},\ \mathbf{e}_1^{(0)}\mathbf{H}') \;\middle|\; \begin{array}{l} \mathbf{e}_1^{(0)} \in \{-1, 0, 1\}^{\frac{n-d}{2}} \times \{\mathbf{0}\}^{\frac{n-d}{2}}, \\ \|\mathbf{e}_1^{(0)}\|_1 = p_1/2 \end{array} \right\}.$$

Similarly to construct $L_2^{(0)}$, enumerate all vectors $\mathbf{e}_2^{(0)}$ non-zero only on the $\{(n-d)/2+1,\ldots,(n-d)\}$ coordinates, with $\|\mathbf{e}_2^{(0)}\|_1 = p_1/2$, and compute $\mathbf{e}_2^{(0)}\mathbf{H}'$.

$$L_2^{(0)} = \left\{ (\mathbf{e}_2^{(0)},\ \mathbf{e}_2^{(0)}\mathbf{H}') \;\middle|\; \begin{array}{l} \mathbf{e}_2^{(0)} \in \{\mathbf{0}\}^{\frac{n-d}{2}} \times \{-1,0,1\}^{\frac{n-d}{2}}, \\ \|\mathbf{e}_2^{(0)}\|_1 = p_1/2 \end{array} \right\}.$$

Level 1. Choose a random vector $R_1 \in \mathbb{Z}_{q-1}^{l_1}$. Run the nearest-neighbour search algorithm (Algorithm 4) on l_1 coordinates to find pairs $(\mathbf{e}_1^{(0)}, \mathbf{e}_2^{(0)})$ such that $(\mathbf{e}_1^{(0)} + \mathbf{e}_2^{(0)})\mathbf{H}' \approx R_1$, meaning there exists some $\mathbf{e}''_{11} \in \{-1,0,1\}^{l_1}$ with $\|\mathbf{e}''_{11}\|_1 = w''_1$ that satisfy

$$(\mathbf{e}_1^{(0)} + \mathbf{e}_2^{(0)})\mathbf{H}' + \mathbf{e}''_{11} = R_1.$$

The vectors $\mathbf{e}_1^{(0)}$ and $\mathbf{e}_2^{(0)}$ add up to a vector $\mathbf{e}_1^{(1)} := \mathbf{e}_1^{(0)} + \mathbf{e}_2^{(0)}$ of weight p_1, producing the list

$$L_1^{(1)} = \left\{ (\mathbf{e}_1^{(1)},\ \mathbf{e}_1^{(1)}\mathbf{H}') \;\middle|\; \begin{array}{l} \mathbf{e}_1^{(1)} \in \{-1,0,1\}^{n-d},\ \|\mathbf{e}_1^{(1)}\|_1 = p_1, \\ \mathbf{e}_1^{(1)}\mathbf{H}' + \mathbf{e}''_{11} = R_1 \end{array} \right\}.$$

An analogous procedure applied to the other two level-0 lists yields $L_2^{(1)}$ where $\mathbf{e}''_{12} \in \{-1,0,1\}^{l_1}$ with $\|\mathbf{e}''_{12}\|_1 = w''_1$.

$$L_2^{(1)} = \left\{ (\mathbf{e}_2^{(1)},\ \mathbf{e}_2^{(1)}\mathbf{H}') \;\middle|\; \begin{array}{l} \mathbf{e}_2^{(1)} \in \{-1,0,1\}^{n-d},\ \|\mathbf{e}_2^{(1)}\|_1 = p_1, \\ \mathbf{e}_2^{(1)}\mathbf{H}' + \mathbf{e}''_{12} = \mathsf{c} - R_1 \end{array} \right\}.$$

Level 2. Finally, use the lists $L_1^{(1)}$ and $L_2^{(1)}$ to merge on the remaining $d - l_1$ coordinates, obtaining vectors of ℓ_1-norm w_2 on these coordinates. This produces the desired list $L_1^{(2)}$ defined at the start. See Figure 5.1 for a clear illustration.

Success probability and correctness.

For the algorithm to succeed there must exist a vector $\mathbf{e}'$ with $\|\mathbf{e}'\|_1 = p$ such that $\mathbf{e}'' = \mathsf{c} - \mathbf{e}'\mathbf{H}' \in \{-1,0,1\}^d$ and the vector $\mathbf{e}''$ is split according to the target weight distribution (induced by the row permutation π). The probability that a random row-permutation induces the desired weight distribution is

$$\Pr(\text{good splitting}) = \frac{\binom{n-d}{p}\binom{l_1}{w_1}\binom{l_2}{w_2}}{\binom{n}{w}}. \tag{5.14}$$

Correctness further requires that the vector $\mathbf{e}'$ admits a decomposition $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$, where $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)} \in \{-1,0,1\}^{n-d}$, and that the corresponding error parts on the l_1 coordinates split

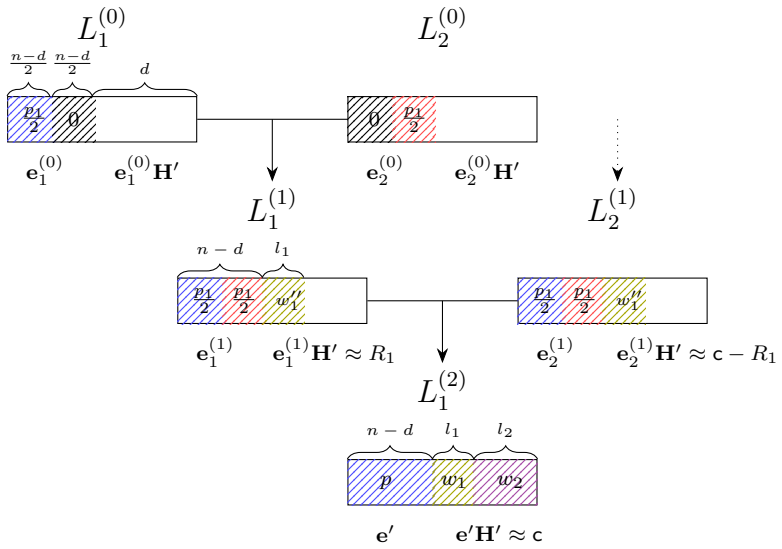


Figure 5.1: Four-lists decoding algorithm.

as $\mathbf{e}_1'' = \mathbf{e}_{11}'' + \mathbf{e}_{12}''$, where $\mathbf{e}_{11}'', \mathbf{e}_{12}'' \in \{-1, 0, 1\}^{l_1}$. For a randomly chosen $R_1 \leftarrow\!\$\ \mathbb{Z}_{q-1}^{l_1}$ the algorithm considers approximate mergings of the form

$$\mathbf{e}_1^{(1)}\mathbf{H}' + \mathbf{e}_{11}'' = R_1 \qquad \text{and} \qquad \mathbf{e}_2^{(1)}\mathbf{H}' + \mathbf{e}_{12}'' = \mathsf{c} - R_1.$$

The number of ways to split $\mathbf{e}'$ into $\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ (with the parameters used in the construction) is

$$\mathcal{N}_1 := 2^{\alpha} \binom{p}{p/2} \binom{n-d-p}{\alpha}, \tag{5.15}$$

and the number of ways to split $\mathbf{e}_1'' \in \{-1, 0, 1\}^{l_1}$ is

$$\mathcal{N}_1' := 2^{\beta} \binom{w_1}{w_1/2} \binom{l_1 - w_1}{\beta}. \tag{5.16}$$

For a uniformly random choice of R_1 we expect at least one valid splitting to exist whenever the total number of candidate split-pairs exceeds the number of possible R_1 values, i.e. when

$$\mathcal{N}_1 \mathcal{N}_1' \ \geq\ (q-1)^{l_1}. \tag{5.17}$$

Thus in practice one picks l_1, α and β so that

$$\frac{\mathcal{N}_1 \mathcal{N}_1'}{(q-1)^{l_1}} \ \geq\ 1,$$

which ensures a valid splitting pair exists with significant probability.

Complexity of the algorithm

The total running time of the algorithm depends on the nearest-neighbour cost and the size of the lists constructed at each level. We use Algorithm 4 (defined in the previous section) and use the hash function to look for the approximate merges between lists.

In the initial level we construct lists directly containing vectors $\mathbf{e}_i^{(0)}$ with ℓ_1-norm $p_1/2$ as explained above. Therefore, the expected size of each list is

$$S_0 := |L_i^{(0)}| = 2^{p_1/2}\binom{(n-d)/2}{p_1/2}.$$

To form the lists in the next level, we merge list pairs from the initial level. We approximately merge elements from $L_1^{(0)}$ and $L_2^{(0)}$ using the NN Algorithm 4 for a randomly chosen $R_1 \leftarrow\$ \mathbb{Z}_{q-1}^{l_1}$. This algorithm requires time

$$T^{(0)} = \mathcal{O}\left(|L_1^{(0)}|\log(|L_1^{(0)}|) + |L_2^{(0)}||\text{Flips}(\mathbf{v})|\log(|L_1^{(0)}|) + |L_1^{(0)}|\cdot|L_2^{(0)}|\frac{|\text{Flips}(\mathbf{v})|}{2^{l_1\nu_1}}\right),$$

where $\mathbf{v} = R_1 - \mathbf{e}_2^{(0)}\mathbf{H}'$ and $|\text{Flips}(\mathbf{v})| = \left(1+\frac{2^{\nu_1+1}}{q-1}\right)^{l_1}$. For hashing we take ν_1 to be the most significant bits of each value in $\mathbb{Z}_{q-1}$ and choose ν_1 to minimize the cost of algorithm.

Let $\mathcal{E}_1$ be the valid error set of $\mathbf{e}_{11}'', \mathbf{e}_{12}''$, which is defined as

$$\mathcal{E}_1 = \left\{\tilde{\mathbf{e}} \,\middle|\, \tilde{\mathbf{e}} \in \{-1,0,1\}^{l_1} \text{ and } \|\tilde{\mathbf{e}}\|_1 = w_1''\right\},$$

and the size of $\mathcal{E}_1$ is

$$|\mathcal{E}_1| = 2^{w_1''}\binom{l_1}{w_1''}.$$

In level 1 we expect the list sizes to be approximately

$$S_1 := |L_i^{(1)}| = \frac{S_0^2\cdot|\mathcal{E}_1|}{(q-1)^{l_1}}.$$

For the final merge we use the level-1 lists and perform the NN merge on $l_2 = d - l_1$ coordinates. The NN algorithm requires time,

$$T^{(1)} = \mathcal{O}\left(|L_1^{(1)}|\log(|L_1^{(1)}|) + |L_2^{(1)}||\text{Flips}(\mathbf{v})|\log(|L_1^{(1)}|) + |L_1^{(1)}|\cdot|L_2^{(1)}|\frac{|\text{Flips}(\mathbf{v})|}{2^{l_2\nu_2}}\right),$$

where for hashing we choose ν_2 to optimise the cost and $\mathbf{v} = \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}'$ and $|\text{Flips}(\mathbf{v})| = \left(1 + \frac{2^{\nu_2+1}}{q-1}\right)^{l_2}$. The permitted error set is

$$\mathcal{E}_2 = \left\{\hat{\mathbf{e}} \,\middle|\, \hat{\mathbf{e}} \in \{-1, 0, 1\}^{d-l_1} \text{ and } \|\hat{\mathbf{e}}\|_1 = w_2\right\}.$$

Recall that $w_2 = w - p - w_1$.

This gives a total expected running time,

$$\text{Running time} = \frac{\max\{S_0,\ S_1,\ T^{(0)},\ T^{(1)}\}}{\Pr[\text{good splitting}]}. \tag{5.18}$$

We use the following parameter choices in our calculations. Take $\|\mathbf{e}\|_1 = w = d - 1$, $\|\mathbf{e}'\|_1 = p = \frac{(n-d)(d-1)}{n}$ and $p_1 = \frac{p}{2} + \alpha$. Let $w_1 = \frac{(d-1)l_1}{n}$, and $w_1/2 + \beta$ is rounded to the nearest integer and set as w_1''. Over several iterations we choose $l_1 = d/9$ for $n = 898$ and $l_1 = d/10$ for $n = 1390, 1840$, which gives the minimum cost, with α and β taken to be the first non-zero entries that satisfy $\mathcal{N}_1\mathcal{N}_1' \geq (q-1)^{l_1}$. Table 5.5 shows our parameter choices for the four-list decoding algorithm. Choosing optimised parameters using optimisation techniques is discussed in [BGLS19].

Parameter choices (n, d, q)	l_1	l_2	p	α	w_1	β	ν_1	ν_2
$(898, 260, 1409)$	29	231	184	16	8	1	9	2
$(1390, 386, 2209)$	39	347	278	20	11	1	10	2
$(1840, 520, 2371)$	52	468	372	27	15	1	10	2

Table 5.5: Parameter choices for the four-list decoding algorithm.

Finally, for the selected parameters, we calculate the running time using Equation (5.18). We observed that the Near Neighbour search is the most expensive step in the calculations. Table 5.6 compares the lower bounds for the costs of ISD attacks in the two-list and four-list variants. Note that our analysis uses simplified assumptions, as our objective is to select secure parameters for the LLXY KEM rather than to compute the exact cost of breaking the scheme. Therefore, we do not claim the existence of a decoding algorithm with the stated complexities. Rather, for any decoding algorithm relying on nearest-neighbour search, the running time will always exceed the values reported here. These values therefore serve as lower bounds for such algorithms. To accommodate potential future improvements, our parameters are chosen with a generous margin of safety against ISD attacks.

Parameter choices (n, d, q)	2 lists lower bound $\log_2$(NN search)	4 lists lower bound $\log_2$(NN search)
$(898, 260, 1409)$	382.842	269.59
$(1390, 386, 2209)$	581.489	445.856
$(1840, 520, 2371)$	768.478	587.642

Table 5.6: Lower bounds for the ISD costs for our parameter choices.

5.3 More general error distributions

For decoding we need the error to satisfy $\|\mathbf{e}\|_1 \leq d - 1$. In theory, the error vector can be sampled anywhere in the ℓ_1 ball in $\mathbb{Z}^n$. LLXY use errors in $\{0, 1\}^n$. Lapiha suggests errors in $\{-1, 0, 1\}^n$. We consider more general subsets of $\mathbb{Z}^n$ with ℓ_1 norm less than d. In this case, one will have to consider the effect on the decoding algorithm and then the cost for lattice and decoding attacks.

Intuitively, the cost for lattice attack increases with higher Euclidean norm. Choosing errors in $\{-1, 0, 1\}^n$ essentially minimizes the ℓ_2-norm and so makes the lattice attacks most effective. Hence, it is worth considering distributions that take values other than $\{-1, 0, 1\}^n$ and so increase the Euclidean norm.

However, the decoding attacks become easier as the Hamming weight of the error vector decreases (if one fixes the ℓ_1-norm and allows larger non-zero entries, then the overall number of non-zero entries decreases). So it is necessary to understand the trade-off between decoding attacks and lattice attacks.

To illuminate this question we choose errors in $\{-2, -1, 0, 1, 2\}^n$ (with an equal number of each). If there are x number of entries from each,

$$\|\mathbf{e}\|_1 = |2x| + |-2x| + |x| + |-x| < d \implies x < \frac{d}{6}$$

This gives Euclidean norm about $1.29\sqrt{d}$.

First, we estimate the cost of the two-list Near Neighbour (NN) ISD attack when the error vector lies in $\{-2, -1, 0, 1, 2\}^n$, with all non-zero values occurring equally often. From the discussion in Section 5.2.1, we recall that the cost of the Near Neighbour attack essentially depends on two components:

- the size of the lists, $|L_1^{(1)}| = |L_2^{(1)}|$, and
- the running time of the Near Neighbour algorithm itself.

For the lattice attacks, we must first adapt the lattice estimator [APS15] to account for the quaternary error distribution. Specifically, we replace the sparse ternary distribution $\{-1,0,1\}$ with the generalised distribution $\{-2,-1,0,1,2\}$ by computing its mean, standard deviation, and density. Once this modified distribution is incorporated into the estimator, we can evaluate the costs of lattice attacks for the generalised errors.

5.3.1 Two-list ISD attack for $\mathbf{e} \in \{-2,-1,0,1,2\}^n$.

We now extend the two-list Near Neighbour search approach to the case where the error vector $\mathbf{e} \in \{-2,-1,0,1,2\}^n$. We follow the same steps as for the $\mathbf{e} \in \{-1,0,1\}^n$ instance. We start by splitting $\mathbf{e} = \begin{pmatrix}\mathbf{e}' & \mathbf{e}''\end{pmatrix}$ and write $\mathbf{e}' = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ where $\mathbf{e}_1^{(1)}$ is non-zero only on the first $(n-d)/2$ coordinates, $\mathbf{e}_2^{(1)}$ is non-zero only on coordinates $\{(n-d)/2+1,\ldots,n-d\}$.

In this analysis, we assume that $\mathbf{e}$ contains exactly u coordinates in $\{\pm 1\}$ and exactly u coordinates in $\{\pm 2\}$, with all non-zero values chosen with equal probability. The ℓ_1-norm of such an error vector is

$$\|\mathbf{e}\|_1 = u \cdot 1 + u \cdot 2 = 3u < d.$$

We also assume that both $\mathbf{e}_1^{(1)}$ and $\mathbf{e}_2^{(1)}$ contain exactly p coordinates in $\{\pm 1\}$, and p coordinates in $\{\pm 2\}$, with the remaining entries being zero. Here

$$p \approx u \cdot \frac{n-d}{2n}$$

represents the expected number of entries of each type in a block of length $(n-d)/2$ under a uniform random permutation. Under this assumption, the last block $\mathbf{e}'' \in \{-2,-1,0,1,2\}^d$ contains $u-2p$ coordinates in $\{\pm 1\}$, and $u-2p$ coordinates in $\{\pm 2\}$.

Equation (5.4) gives

$$\mathbf{e}_1^{(1)}\mathbf{H}' + \mathbf{e}_2^{(1)}\mathbf{H}' + \mathbf{e}'' = \mathsf{c} \quad \Longrightarrow \quad \mathbf{e}_1^{(1)}\mathbf{H}' \approx \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}'.$$

Now the Near Neighbour component is $\mathbf{e}'' \in \{\pm 2, \pm 1, 0\}^d$ and the set of possible values of $\mathbf{e}''$ is

$$\mathcal{E} = \{\mathbf{x} \in \{\pm 2, \pm 1, 0\}^d \mid \#\{i : x_i = \pm 2\} = \#\{i : x_i = \pm 1\} = u - 2p\},$$

where $|\mathcal{E}| = 2^{2u-4p}\binom{d}{u-2p}\binom{d-(u-2p)}{u-2p}$. Thus, finding an approximate merge $\mathbf{e}_1^{(1)}\mathbf{H}' \approx \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}'$ is equivalent to checking whether

$$\mathbf{e}'' = \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' - \mathbf{e}_1^{(1)}\mathbf{H}' \in \mathcal{E}.$$

To find an approximate match, two lists are constructed as follows:

$$L_1^{(1)} = \left\{ \left(\mathbf{e}_1^{(1)}\mathbf{H}', \mathbf{e}_1^{(1)}\right) \;\middle|\; \begin{array}{l} \mathbf{e}_1^{(1)}\mathbf{H}' \in \mathbb{Z}_{q-1}^d, \quad \mathbf{e}_1^{(1)} \in \{\pm 2, \pm 1, 0\}^{\frac{n-d}{2}} \times \{\mathbf{0}\}^{\frac{n-d}{2}}, \\ \#\{i : \mathbf{e}_{1,i}^{(1)} = \pm 2\} = \#\{i : \mathbf{e}_{1,i}^{(1)} = \pm 1\} = p \end{array} \right\},$$

$$L_2^{(1)} = \left\{ \left(\mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}', \mathbf{e}_2^{(1)}\right) \;\middle|\; \begin{array}{l} \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}' \in \mathbb{Z}_{q-1}^d, \quad \mathbf{e}_2^{(1)} \in \{\mathbf{0}\}^{\frac{n-d}{2}} \times \{\pm 2, \pm 1, 0\}^{\frac{n-d}{2}}, \\ \#\{i : \mathbf{e}_{2,i}^{(1)} = \pm 2\} = \#\{i : \mathbf{e}_{2,i}^{(1)} = \pm 1\} = p \end{array} \right\}. \tag{5.19}$$

The number of possible vectors $\mathbf{e}_1^{(1)}$ of desired shape is obtained by first choosing p positions for the ± 1 entries in the block, which can be done in $\binom{(n-d)/2}{p}$ ways. From the remaining $(n-d)/2 - p$ coordinates, we then choose p positions for the ± 2 entries, giving $\binom{(n-d)/2-p}{p}$ possibilities. Each ± 1 position can take two possible signs, as can each ± 2 position, contributing a factor $2^p \cdot 2^p = 2^{2p}$. Therefore, the size of each list is

$$|L_1^{(1)}| = |L_2^{(1)}| = 2^{2p}\binom{\frac{n-d}{2}}{p}\binom{\frac{n-d}{2}-p}{p}.$$

As we discuss in the previous section, the construction of lists $L_1^{(1)}$ and $L_2^{(1)}$ and the existence of $\mathbf{e}''$ with the correct weight distribution is guaranteed by the probabilistic assumptions that the vector $\mathbf{e}$ had the correct distribution in the first place. The total number of valid vectors $\mathbf{e} \in \{\pm 2, \pm 1, 0\}^n$ with exactly u number of ± 1's and u number of ± 2's is:

$$N_{\text{total}} = \binom{n}{u}\binom{n-u}{u} \cdot 2^{2u}.$$

The number of "good-split" vectors, i.e., those with the desired distribution in $\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)}$ and $\mathbf{e}''$, is:

$$N_{\text{good}} = \left[\binom{\frac{n-d}{2}}{p}\binom{\frac{n-d}{2}-p}{p}\right]^2 \cdot \binom{d}{u-2p}\binom{d-(u-2p)}{u-2p}2^{2u}.$$

Hence, the probability of getting a good split is

$$\Pr(\text{good split}) = \frac{\left[\binom{\frac{n-d}{2}}{p}\binom{\frac{n-d}{2}-p}{p}\right]^2 \cdot \binom{d}{u-2p} \cdot \binom{d-(u-2p)}{u-2p}}{\binom{n}{u} \cdot \binom{n-u}{u}}.$$

The expected number of repetitions until a good split occurs is $1/\Pr(\text{good split})$.

In each repetition, we generate the two uniform and independent lists $L_1^{(1)}$ and $L_2^{(1)}$ as in Equation (5.19) and use the Near Neighbour algorithm given in Algorithm 4 with inputs $L_1^{(1)}$ and $L_2^{(1)}$ to find and output a list $L_1^{(2)}$ containing $\mathbf{e}$ of correct weight distribution. Using

hash collisions, the NN Algorithm has a cost of

$$\text{Cost of NN Algorithm} = \mathcal{O}\left(\begin{array}{c} |L_1^{(1)}| \log |L_1^{(1)}| + |L_2^{(1)}||\text{Flips}(\mathbf{v})| \log |L_1^{(1)}| \\ +|L_1^{(1)}|\,|L_2^{(1)}| \frac{|\text{Flips}(\mathbf{v})|}{2^{d\nu}} \end{array} \right).$$

Note that when $\mathbf{e} \in \{-2,-1,0,1,2\}$, the number of "borderline" entries become $2^{\nu+2}$ since now we have to account for four cases; $F(a) \neq F(a+1)$, $F(a) \neq F(a-1)$, $F(a) \neq F(a+2)$ or $F(a) \neq F(a-2)$. Therefore $|\text{Flips}(\mathbf{v})| = \left(1 + \frac{2^{\nu+2}}{q-1}\right)^d$ where $\mathbf{v} = \mathsf{c} - \mathbf{e}_2^{(1)}\mathbf{H}'$.

The average cost per repetition is therefore,

$$T = \max\{\text{List sizes}, \text{Cost of NN Algorithm}\}.$$

So the total average cost is:

$$\text{Average ISD cost} = \frac{T}{\Pr(\text{good split})}. \tag{5.20}$$

Our experiments indicate that, the decoding attacks experience only a small security degradation under this distribution, and therefore there remains significant potential to generalise the error model while preserving strong security guarantees.

5.3.2 Lattice attacks for $\mathbf{e} \in \{-2,-1,0,1,2\}^n$

Recall that our LWE instance is $(\mathbf{H}', \mathsf{c})$, where $\mathsf{c} = \mathbf{e}'\mathbf{H}' + \mathbf{e}''$. Here $\mathbf{e}'$ is the LWE secret of length $(n-d)$ and $\mathbf{e}''$ is the LWE error of length d, both derived from the same vector $\mathbf{e}$ sampled from a sparse, low-norm distribution. In this setting, we now assume $\mathbf{e} \in \{-2,-1,0,1,2\}^n$ with ℓ_1-norm bounded by $\|\mathbf{e}\|_1 < d$.

To compute the cost of lattice attacks when $\mathbf{e} \in \{-2,-1,0,1,2\}^n$, we use the lattice estimator [APS15] with a minor modification. The estimator models distributions of vectors of length n with p entries equal to 1 and m entries equal to -1 using the function $\mathsf{SparseTernary}()$. We extend this by defining a new function, $\mathsf{SparseQuaternary}()$, which outputs the distribution of vectors of length n with p entries of 1, m entries of -1, x entries of 2, y entries of -2, and the remaining coordinates equal to 0.

Following the same conventions as in the decoding attack analysis, we assume that $\mathbf{e}$ contains exactly u coordinates in $\{\pm 1\}$ and exactly u coordinates in $\{\pm 2\}$, with each non-zero value chosen uniformly at random. In our calculations we set $u = \left\lfloor \frac{d-1}{3} \right\rfloor$, and assume that the $\{\pm 1\}$ entries are split evenly between $+1$ and -1, and likewise the $\{\pm 2\}$ entries are

split evenly between $+2$ and -2. Consequently, in $\mathsf{SparseQuaternary}()$ we have $p = m = x = y$.

Once the counts of $\{\pm 1\}$ and $\{\pm 2\}$ entries in the secret and error vectors of the LWE instance $(\mathbf{H}', \mathsf{c})$ are fixed according to this distribution, we estimate the cost of lattice attacks using the modified estimator. As expected, the cost of lattice attacks is higher when $\mathbf{e} \in \{-2, -1, 0, 1, 2\}^n$ compared to the case $\mathbf{e} \in \{-1, 0, 1\}^n$.

Candidate parameters for both lattice-based and decoding attacks, under this generalised error distribution, are presented in Section 5.4.2. Owing to the small security loss from decoding attacks and the substantial security gain from lattice attacks, exploring error distributions beyond the classical $\{0, 1\}$ and $\{-1, 0, 1\}$ cases appears promising for achieving a more balanced trade-off between lattice and ISD attack complexities when selecting parameters for the LLXY KEM.

5.4 Parameters / Other optimisations

5.4.1 Optimising q versus d

The LLXY paper restricts to $\alpha_i, \beta_j \in \mathbb{F}_q$ and so requires $q > n + d$. But one can replace $(x - \alpha_i)$ by any irreducible polynomial $a_i(x)$. So suppose we work with irreducible quadratic polynomials $a_i(x) = x^2 + e_i x + f_i \in \mathbb{F}_q[x]$ where $e_i^2 - 4f_i$ is not a square in $\mathbb{F}_q$. Note that $\mathbb{F}_q[x]/(a_i(x)) \cong \mathbb{F}_{q^2}$. There are about $q^2/2$ such irreducible polynomials. For decoding d errors in a code using quadratic polynomials we now need $c(x)$ to be of degree $2d$. Hence, as long as $n < d^2/2$, we have reduced q from $q > n + d$ to $q' > d$, at the expense of increasing the lattice volume to $((q')^2 - 1)^d \approx (q')^{2d}$.

Note that the ciphertext size is proportional to $\log_2(q^d)$. So how does q^d compare with $(q')^{2d}$? If $n + d \approx d^2$ (which is not likely in practice, but it gives the idea) then $q \approx (q')^2$ and so $q^d \approx (q')^{2d}$ and the ciphertexts are the same size.

This shows that we can trade-off the size of the modulus q against the degree of the polynomial $c(x)$, by not requiring all $\alpha_i \in \mathbb{F}_q$, and this may not have much effect on the ciphertext size.

In the extreme case one could take $a_i(x)$ and $c(x)$ all irreducibles of some degree t, and then one could choose q as small as possible given the constraint that the number of irreducibles of degree t is at least $n + d$, roughly $n + d < q^t/t$. In the extreme one could even take $q = 2$.

5.4.2 Parameters with Estimated Costs

In this section, we present candidate parameter sets for the LLXY KEM that achieve the desired post-quantum security levels. The analysis evaluates the hardness of the lattice attacks using the LWE estimator [APS15], which summarises the cost of advanced lattice-based attacks. We also estimate the resistance to decoding attacks using our experiments with the Nearest Neighbour information set decoding (ISD) algorithms, focusing on 2-list and 4-list variants.

For a fixed value of n, we perform a search over a suitable range of error vector weights d, typically between $n/4 \lesssim d \lesssim n/2$, ensuring the error vector is sparse but non-trivial. The modulus q is chosen as a prime slightly larger than $n + d$. We first consider ternary error vectors with entries from the set $\{-1, 0, 1\}^n$. This choice keeps the Euclidean norm of the error vector relatively small, which is advantageous in terms of decoding efficiency.

Security Estimates for Ternary Error Vectors. Table 5.7 presents the selected parameter sets and their associated security estimates. We choose the lattice attack with the minimum cost and calculated a lower bound for the decoding attacks using 2-list and 4-list NN ISD algorithms. In every instance we examined, lattice attacks outperformed decoding attacks in the LLXY setting.

Parameter choices			$\log_2$(LWE estimator)	2 lists	4 lists
n	d	q		lower bound $\log_2$(NN search)	lower bound $\log_2$(NN search)
898	260	1409	140.2	382.842	296.59
1390	386	2209	199.6	581.489	445.856
1840	520	2371	262.1	768.478	587.642

Table 5.7: Estimated security levels for LLXY KEM when $\mathbf{e} \in \{-1, 0, 1\}^n$.

Remark 5.1. *The original LLXY paper [LLXY20] only analysed the unique-SVP attack. However, our LWE estimator results show that dual and hybrid attacks are more effective and should be considered in parameter selection.*

Security Analysis with General Error Distributions. In Section 5.3, we consider an alternative error distribution: vectors drawn from $\{-2, -1, 0, 1, 2\}^n$, where each non-zero value appears with equal frequency. This increases the Euclidean norm of the error vector, which typically improves resistance to lattice attacks. However, it also reduces the Hamming weight from d to approximately $(2/3)d$, which can make decoding attacks more effective.

Table 5.8 summarises the estimated attack costs under this general error distribution. We only considered the two-list version of the ISD attack in these calculations.

n	d	q	$\log_2$(LWE estimator)	lower bound $\log_2$(NN-ISD)
898	260	1409	162.6	369.097
1390	386	2209	242.2	548.313
1840	520	2371	317.7	729.213

Table 5.8: Estimated security levels for LLXY KEM when $\mathbf{e} \in \{-2, -1, 0, 1, 2\}^n$.

This trade-off suggests a slight security loss against decoding attacks, which is offset by a comparable security gain from lattice attacks for the same parameter sets. Therefore, we believe that generalising the error distribution offers a promising approach to balancing security and efficiency.

For example, with parameters $n = 720, d = 180, q = 997$, we obtain Lattice attack cost (estimated via the LWE estimator) is 2^{141}, while the NN-ISD lower bound is $2^{287.195}$. However, these same parameters become *insecure* if the error distribution is limited to $\{-1, 0, 1\}^n$: in this case, the hybrid lattice attack has a cost of only 2^{104}, which is below the standard security threshold.

Key and Ciphertext Sizes. We also evaluate the efficiency of the LLXY KEM by computing the sizes of the public key, secret key, and ciphertext. The scheme has the following structure:

- Public key (pk): Matrix $\mathbf{H}' \in \mathbb{Z}_{q-1}^{(n-d)\times d}$, size $(n-d)d \cdot \lceil \log_2(q-1) \rceil$ bits.
- Secret key (sk): Includes a degree-d polynomial and n elements in $\mathbb{Z}_{q-1}$, total size $(n+d) \cdot \lceil \log_2(q-1) \rceil$ bits.
- Ciphertext (c): Vector in $\mathbb{Z}_{q-1}^d$, size $d \cdot \lceil \log_2(q-1) \rceil$ bits.

We observed that the LLXY KEM provides better public key sizes compared to the classic McEliece system [BCL+17], albeit at the cost of larger ciphertexts. Specifically, in the classic McEliece system, ciphertext sizes range from 128 to 240 bytes, with public key sizes varying between 0.26 and 1.35 megabytes. In comparison, the LLXY KEM scheme achieves smaller public keys, with a maximum size of 0.95 megabytes, but at the expense of larger ciphertexts. Moreover, the BIKE [ABB+22] and HQC [MAB+18] schemes have the advantage of shorter public keys but comparatively large cipher texts. We observe that the ciphertext sizes of our scheme are substantially smaller than that of BIKE and HQC, but

our public keys are larger. Table 5.9 shows the public key and ciphertext sizes (in bytes) of NIST candidate code-based schemes, BIKE, HQC, and Classic McEliece targeting security level 3.

	Public key size (bytes)	Ciphertext size (bytes)
McEliece	524,160	188
BIKE	3082	3114
HQC	4522	9042

Table 5.9: Public key and ciphertext sizes of code-based schemes (NIST Level 3)

As a final note, Table 5.10 compares LLXY KEM with CRYSTALS-Kyber at equivalent security levels. While Kyber offers compact key and ciphertext sizes, LLXY achieves notably smaller ciphertexts, albeit with larger public keys.

	LWE estimator	Public key (bytes)	Ciphertext (bytes)	LLXY KEM (n, d, q)	LWE estimator	Public key (bytes)	Ciphertext (bytes)
Kyber-512	139.7	800	768	$(898, 260, 1409)$	140.2	207,350	325
Kyber-768	196.4	1184	1088	$(1390, 386, 2207)$	199.6	532,873	531
Kyber-1024	262.4	1568	1568	$(1840, 520, 2371)$	262.1	943,800	715

Table 5.10: Comparison of Kyber and LLXY KEM ciphertext and public key sizes.

In summary, the analysis of parameter sets for the LLXY KEM highlights several key findings:

- Dual and hybrid lattice attacks outperform the embedding attack considered in the original LLXY in terms of effectiveness, and thus serve as a better guide for selecting secure parameters.
- For the LLXY scheme, lattice-based attacks remain more effective than decoding attacks even when the error vectors are sampled from sparse sets such as $\{-1, 0, 1\}^n$ or slightly wider ones like $\{-2, -1, 0, 1, 2\}^n$.
- Using generalised error vectors (e.g., $\{-2, -1, 0, 1, 2\}$) improves resistance to lattice attacks but mildly weakens decoding resistance. This suggests that there's room to consider more general errors.
- The main drawback is the large public key size, which is common to most of the code-based schemes. Investigating techniques to reduce key sizes without compromising security would be a valuable direction for future research.

5.5 Summary & Future Directions

A comprehensive security analysis of the modified scheme was conducted, incorporating the most recent advances in both lattice-based and Information Set Decoding (ISD) attacks. This analysis includes parameter evaluations and security level estimations, taking into account dual, hybrid, and decoding attacks, and providing guidance on safe parameter selection.

Our experimental results suggest that the scheme offers flexibility in generalising the error distribution in terms of balancing efficiency and security. While the current construction relies on ternary error vectors (i.e., elements from $\{-1, 0, 1\}$), our analysis shows potential for extending to more general distributions. This opens the door to further optimisations, both in decoding efficiency and in resistance to lattice attacks. As part of future work, we recommend a more systematic investigation into alternative error distributions, which may lead to improved trade-off between lattice and decoding attack security margins.

Additionally, we emphasise the need for deeper cryptanalytic study of the scheme's underlying hardness assumption. Since the public key corresponds to a family of lattices with a special secret decoding algorithm, it is crucial to evaluate whether this structure admits previously unexplored vulnerabilities or specialised attacks.

Overall, we believe this section of the thesis establishes a solid foundation for the LLXY scheme, both from a theoretical and practical standpoint, and highlights promising avenues for continued development and cryptanalysis.

Part II

A Study on New k-tree Algorithms.

Chapter 6

On k-sum Problem for $\{-1, 1\}^m$ Vectors

Contents

In this chapter, we study a natural and seemingly unexplored computational problem: Given k lists of vectors in $\{-1, 1\}^m$, find a selection of one vector from each list such that their sum is equal to the zero vector.

This problem is intuitively appealing. The sum of k vectors sampled uniformly at random from $\{-1, 1\}^m$ behaves like a random walk in $\mathbb{Z}^m$, and when k is even, the distribution of each coordinate of the resulting sum is centred at zero. Due to the Central Limit Theorem, the coordinates of such a sum tend to concentrate within an interval of size $\mathcal{O}\left(\sqrt{k}\right)$, making the zero vector not only a plausible target but also the most probable

outcome in each coordinate. Despite this natural structure, no prior work appears to have studied the complexity or algorithmic solutions to this problem.

Our interest in this problem is motivated by an attempted cryptanalysis of a newly proposed computational assumption. Specifically, the problem arises in the context of analysing a variant of the Inhomogeneous Short Integer Solution (ISIS) problem introduced by Baldimtsi, Cheng, Goyal, and Yadav [BCGY24]. A more detailed discussion of their scheme and our cryptanalytic observations will follow in a later chapter.

We undertake a detailed study of this problem in this chapter, providing a comprehensive analysis of its computational complexity. For clarity and consistency, we assume throughout this chapter that all vectors under consideration are of dimension m. In the complexity analysis, we ignore the factors that are logarithmic in the running time unless otherwise mentioned.

6.1 Background and Motivation

One of the most well-known combinatorial tools in cryptography is the birthday problem, as defined in the preliminary chapter (Definition 2.1). The birthday problem is well studied, and a solution can be found with high probability when the sizes of the two lists L_1 and L_2 satisfies the condition $|L_1| \times |L_2| \gg 2^m$. When the list sizes are appropriately chosen, the optimal algorithm has a time complexity of $\Theta\left(2^{m/2}\right)$ up to logarithmic factors of m.

The structure of the birthday problem naturally suggests studying a generalisation involving an arbitrary number of lists. This leads to the k-dimensional extension known as the *k-sum problem*, defined as follows: Given k lists of n-bit values, find a selection of one element from each list such that their bitwise XOR is zero. For $k = 2$, this reduces to the classical birthday problem. The formal definition of the k-sum problem, as originally stated, is:

Definition 6.1 (k-sum problem)**.** *Given k lists $L_1, \ldots, L_k$ of elements drawn uniformly and independently at random from $\{0,1\}^m$, find $\mathbf{x}_1 \in L_1, \ldots, \mathbf{x}_k \in L_k$ such that*

$$\mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \cdots \oplus \mathbf{x}_k = \mathbf{0}.$$

Here, the symbol $\oplus$ denotes the bitwise exclusive-or (XOR) operation, which corresponds to addition modulo 2 for binary vectors. Throughout this chapter, we focus on the case where $k = 2^q$ for some integer q.

An information theoretic lower bound of the k-sum problem over all groups is given by the following theorem [Wag02].

Theorem 6.1. *Let $L_1, \ldots, L_k \subseteq \{0,1\}^m$ be independent random lists of size N. Then a solution to $x_1 \oplus \cdots \oplus x_k = 0$ exists with non-negligible probability only if $N = \Omega(2^{m/k})$. Consequently, any generic algorithm for solving random k-sum instances with non-negligible success probability requires time at least $\Omega(2^{m/k})$, up to polynomial factors.*

Proof. Let $L_1, \ldots, L_k \subseteq \{0,1\}^m$ be random lists of size N, and the goal is to find elements $\mathbf{x}_1 \in L_1, \ldots, \mathbf{x}_k \in L_k$ such that $\mathbf{x}_1 \oplus \cdots \oplus \mathbf{x}_k = \mathbf{0}$. For any such k-tuple, the XOR is uniformly distributed over $\{0,1\}^m$, so the probability that a fixed tuple satisfies the equation is $1/2^m$. There are N^k such tuples in total, so the expected number of solutions is $N^k/2^m$. If $N^k \ll 2^m$, then this expected value is much less than 1, and with high probability, no solution exists. Therefore, to even expect a solution to exist, we must have $N^k = \Omega(2^m)$, implying $N = \Omega(2^{m/k})$. Hence, any algorithm must examine at least $\Omega(2^{m/k})$ possibilities, up to polynomial factors, establishing the lower bound. □

By Theorem 6.1 we know that a solution to the k-sum problem exists with good probability if $\prod_{i=1}^{k} |L_i| > 2^m$ for sufficiently random L_i.

A simple approach to solving this problem, using the birthday paradox (Theorem 2.1) is as follows. First, compute a list $S_1 = \{\mathbf{x}_1 \oplus \cdots \oplus \mathbf{x}_{k/2} \ : \mathbf{x}_i \in L_i\}$, and another list $S_2 = \{\mathbf{x}_{k/2+1} \oplus \cdots \oplus \mathbf{x}_k \ : \mathbf{x}_i \in L_i\}$. If a vector appears in both S_1 and S_2, it provides a solution to the problem, and such a match can be found in time proportional to the sizes of S_1 and S_2. To ensure a high probability of success, we need to make it likely that S_1 and S_2 share at least one common element. By the birthday paradox, this occurs with good probability when the sizes of both lists are $\Theta\big(2^{m/2}\big)$, leading to an overall running time of $\tilde{O}(2^{m/2})$. The attempts to break the square-root complexity barrier opened up a new path of research that is focused around k-sum algorithms.

6.1.1 Existing k-sum algorithms

Camion and Patarin [CP91] are credited with introducing the idea of using tree-based algorithms to solve k-sum problems more efficiently than the generic square-root time approach. Their work focuses on the following problem: given a 128-bit integer b and a set of 256 integers a_i, each 120 bits in length, find a subset of the a_i's that sums to b. They demonstrated how to solve this problem in approximately 2^{32} operations using what is essentially a 4-tree algorithm. This technique was then applied to attack a knapsack-based hash function.

Wagner's k-tree algorithm

Wagner [Wag02] extended their work by considering the general case for k-sum problem for arbitrary groups, by giving the asymptotic complexity of the algorithm and by introducing new applications to cryptanalysis. Wagner's k-tree (aka k-sum, or k-lists) algorithm is an elegant and simple algorithm that solves k-dimensional generalisation of the birthday problem. At the expense of larger lists sizes, they give a cube-root algorithm for $k = 4$ and further improvements when $k > 4$. The algorithm works only when the initial lists are large enough, i.e., in the special case where there are sufficiently many solutions to the k-sum problem. They show that if the initial list length is greater than $2^{m/1+\lfloor \log k \rfloor}$ a solution exists with high probability. So, if lists of size $\mathcal{O}\big(2^{m/(1+\lfloor \log k \rfloor)}\big)$ are used, the k-sum problem can be solved heuristically in $\mathcal{O}\big(k \cdot 2^{m/(1+\lfloor \log k \rfloor)}\big)$ time and space and the algorithm that achieves this is called the k-tree algorithm.

To illustrate the main idea behind the algorithm consider the $k = 4$ case in Figure 6.1. Let $L_1, \dots L_4$ be four lists of length $m_0 = 2^{m/3}$ each. It is necessary to assume that $|L_i| \geq 2^{m/(q+1)}$ for $i = 1, \dots, 4$ and $q = 2$. The algorithm works in two levels by merging the lists in pairs; first, it combines 2 lists L_1 and L_2 to form a list $L_{1,2}$ of pairwise sums (or XORs) that agree on a specific set of coordinates (say, the first $\ell = m/3$ coordinates). Similarly, compute the list $L_{3,4}$ from L_3 and L_4.

$$L_{i,i+1} = \{(\mathbf{x}_i, \mathbf{x}_{i+1}) \in L_j \times L_{i+1} \ : \ (\mathbf{x}_i \oplus \mathbf{x}_{i+1})_\ell = \mathbf{0}\}$$

for $i = 1, 3$. The notation $(\mathbf{x})_\ell = \mathbf{0}$ denotes that a particular block of ℓ coordinates of the vector $\mathbf{x}$ are zero. In this step it is taken as the first $\ell = m/3$ coordinates are zero. These sets can be formed efficiently. For example, to construct $L_{1,2}$, sort the lists L_1, then for each $\mathbf{x}_2 \in L_2$ check whether there exists $\mathbf{x}_1 \in L_2$ such that $(\mathbf{x}_1)_\ell = (\mathbf{x}_2)_\ell$. Then $(\mathbf{x}_1 \oplus \mathbf{x}_2)_\ell = \mathbf{0}$ by definition. If the lists are sufficiently random, then by the linearity of expectation one can consider the expected size of $L_{j,j+1}$ is $|L_j||L_{j+1}| \cdot 2^{-\ell}$.

The second step is to find $(\mathbf{x}_1, \mathbf{x}_2) \in L_{1,2}$ and $(\mathbf{x}_3, \mathbf{x}_4) \in L_{3,4}$ such that $\mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \mathbf{x}_3 \oplus \mathbf{x}_4 = \mathbf{0}$. Since any sum of elements in $L_{1,2}$ and $L_{3,4}$ will be zero on the first $\ell = m/3$ bits, this is done by sorting and searching the remaining $m - \ell$ bits. Moreover, since $|L_{1,2}|, |L_{3,4}| \approx 2^\ell$ and $m - \ell \approx 2\ell$, if the initial lists are sufficiently random, we expect the algorithm to find a solution in time and space complexity $\mathcal{O}\big(2^{m/3}\big)$.

A major limitation of Wagner's algorithm is that it fails when $m_0 \geq 2^{m/1+\lfloor \log k \rfloor}$ doesn't hold. As a result, Wagner's algorithm is only applicable when the input lists meet this requirement, or when it is possible to increase either the length of the lists or the number of lists. However, in some applications, both the number of lists (k) and the list lengths (m_0)

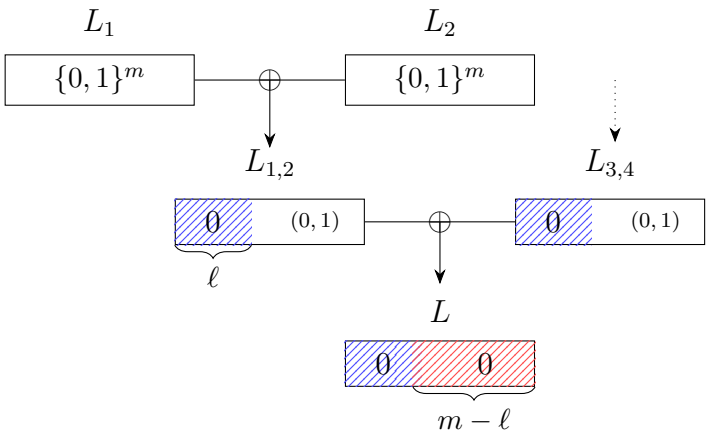


Figure 6.1: Wagner's 4-lists algorithm for binary vectors

are fixed, and m_0 may be smaller than the bound required by Wagner's method. Minder and Sinclair addressed this issue in their work [MS12].

Minder & Sinclair's extended algorithm

Minder and Sinclair [MS12] extended the k-tree algorithm to work for significantly smaller list lengths and gives the first rigorous bound on the failure probability of these k-tree algorithms.

Assume $k = 2^q$ and the initial list length is m_0 as before. Minder and Sinclair address the problem of finding a solution to the k-tree algorithm when q, m_0 and m are fixed, subjected to the non-trivial requirement of $m_0 \geq 2^{m/2^q}$ given by the birthday problem. In particular, they extended k-tree algorithm for

$$2^{m/2^q} \leq m_0 \leq 2^{m/(q+1)}.$$

This algorithm can be seen as interpolating between Wagner's k-tree algorithm and the naïve algorithm: at one extreme ($m_0 = 2^{m/(q+1)}$) it becomes the k-tree algorithm, and at the other ($m_0 = 2^{m/2^q}$) it becomes the classic birthday algorithm. The idea behind the modification is that they vary the number of bits eliminated (i.e., finding vectors that sum to zero) in each round, in such a way that ultimately more bits can be eliminated in total. In the original k-tree algorithm by Wagner, a fixed number $\ell = \frac{m}{q+1}$ of bits were eliminated in each round (except for the last round, where 2ℓ bits are eliminated) to keep the list lengths constant over all rounds and thereby balancing the work done in each round. In [MS12] they design the algorithm in two phases, such that in the first phase simply increase the pool of vectors (by summing combinations from the original lists) until the number of vectors is

large enough for the elimination phase to work successfully. They use integer programming to find an optimal set of parameters that minimizes the running time while guaranteeing a solution exists. Refer Theorem 3.1 of [MS12].

Over the years, the k-sum algorithm has found different variations. As examples, Shallue [Sha08] and Joux et al. [JKL24] have independently analysed the k-list algorithm over a field $\mathbb{Z}_p$ and [BM17a] define the "Approximate k- list problem" where the authors have relaxed the original problem and allows finding a set of vectors that sum to a vector of small Hamming weight. There is also a large literature around the natural connection between the k-sum problem over $(\mathbb{Z}/m\mathbb{Z}, +)$ and the subset sum problem over $\mathbb{Z}/m\mathbb{Z}$ which is out of scope for this thesis.

Due to its effectiveness and simplicity, Wagner's algorithm and its extensions [MS12, NS15, Din19], have become a widely used tool in cryptography and related fields [Lyu05b, MO15, BM18]. Wagner shows a number of cases where the k-sum problem has applications to cryptanalysis of various systems including how to break various blind signature schemes, how to break certain pseudorandom number generators and more. Coron and Joux [CJ04] used Wagner's algorithm to break a hash function based on error correcting codes. Lyubashevsky [Lyu05a] noted that Wagner's algorithm can be applied to solve high density subset-sum problems, and Shallue [Sha08] extended the results.

A more recent application is [DEL25], where the authors prove that a variant of Wagner's algorithm solves SIS with n equations modulo $q = \text{poly}(n)$, in sub-exponential time $\exp(\mathcal{O}(n/\log\log n))$ for an $\ell_\infty-$norm bound $\beta = q/\text{poly}\log(n)$ and only requiring $m = n + \omega(n/\log\log n)$. This variant of SIS underlies the security of the Dilithium scheme which is a candidate submitted to the NIST post-quantum cryptography standardisation. Despite its sub-exponential complexity, Wagner's algorithm does not appear to threaten Dilithium's concrete security.

6.1.2 Motivation and the Research Problem

Generalisations of Wagner's algorithm have been explored in several directions. To the best of our knowledge, replacing binary vectors with vectors in $\{-1,1\}^m$ constitutes a novel variant of the k-sum problem.

In this work, we study how Wagner's algorithm can be adapted to solve a new variation of the generalised birthday problem in which the input vectors are drawn from $\{-1,1\}^m$. Our objective is to analyse the complexity of a k-tree algorithm for this setting and to determine the optimal parameters that minimize the algorithm's running time.

We define the new version of the generalised birthday problem as follows:

Definition 6.2. *Given k lists $L_1, \ldots, L_k$ of vectors drawn uniformly and independently at random from $\{-1,1\}^m$, find $\mathbf{x}_1 \in L_1, \ldots, \mathbf{x}_k \in L_k$ such that $\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_k = \mathbf{0}$.*

Note that in this setting, we no longer work modulo 2 as in the binary case; instead, all computations are carried out over the integers. Intuitively, the ± 1 variant of the generalised birthday problem is more challenging than its binary counterpart. This is because the binary version can be embedded into the ± 1 setting via a reduction modulo 4.

Lemma 6.1. *It is possible to convert the binary problem into a $\{-1,1\}^m$ problem when k is a multiple of 4.*

Proof. Let $k = 4k'$ for some integer k', and consider the map $f : \{0,1\}^m \to \{-1,1\}^m$ defined by $f(\mathbf{x}) = 2\mathbf{x} - \mathbf{1}$, which maps $0 \mapsto -1$ and $1 \mapsto 1$. For any $\mathbf{x} \in \{0,1\}^m$, we have $f(\mathbf{x}) \in \{-1,1\}^m$. Conversely, given $f(\mathbf{x}) = \mathbf{y} \in \{-1,1\}^m$, we can recover $\mathbf{x}$ by computing

$$\mathbf{x} = \frac{\mathbf{y} + \mathbf{1}}{2} \in \{0,1\}^m.$$

Now suppose one solves the $\{-1,1\}^m$ version of the generalised birthday problem. That is, we have found $\mathbf{y}_i \in L_i$ such that $\mathbf{y}_1 + \mathbf{y}_2 + \cdots + \mathbf{y}_k = \mathbf{0}$. Define $\mathbf{x}_i = \dfrac{\mathbf{y}_i + \mathbf{1}}{2}$ for each i, so that $\mathbf{x}_i \in \{0,1\}^m$. Then

$$\sum_{i=1}^{k} \mathbf{x}_i = \sum_{i=1}^{k} \frac{\mathbf{y}_i + \mathbf{1}}{2} = \frac{1}{2}\left(\sum_{i=1}^{k} \mathbf{y}_i + k \cdot \mathbf{1}\right) = \frac{k}{2} \cdot \mathbf{1}.$$

Since $\sum_{i=1}^{k} \mathbf{y}_i = \mathbf{0}$ and $k = 4k'$, it follows that $\frac{k}{2}$ is even, so

$$\sum_{i=1}^{k} \mathbf{x}_i = \mathbf{0} \pmod 2.$$

In other words, $\mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \cdots \oplus \mathbf{x}_k = \mathbf{0}$, where $\oplus$ denotes bitwise XOR. This shows that a solution to the $\{-1,1\}^m$ problem yields a solution to the binary generalised birthday problem whenever k is a multiple of 4. □

For the two-list variant, the $\{-1,1\}^m$ and binary case $\{0,1\}^m$ are essentially identical. In both, the solution reduces to matching elements in a space of size 2^m, so the birthday bound gives lists of size $\Theta(2^{m/2})$ and time complexity $\Theta(2^{m/2})$. The difference appears when $k \geq 4$. To give intuition about how our problem (Definition 6.2) is different from Wagner's original binary XOR problem, we discuss naive (pre-Wagner) ways to solve the 4-list problem using the birthday paradox.

Suppose one is given 4 lists L_1, L_2, L_3, L_4 of binary strings in $\{0,1\}^m$ of the same size, and wants to find $\mathbf{x}_i \in L_i$ such that $\mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \mathbf{x}_3 \oplus \mathbf{x}_4 = \mathbf{0}$. One can expect a solution as soon as $|L_i| = 2^{m/4+o(1)}$. If one had never heard of Wagner's algorithm, but knew about baby-step-giant-step, then the following algorithm would be natural: Compute $L_{12} = \{(\mathbf{x}_1 \oplus \mathbf{x}_2, \mathbf{x}_1, \mathbf{x}_2) : \mathbf{x}_1 \in L_1, \mathbf{x}_2 \in L_2\}$ and sort it with respect to the first coordinate. Then we expect $|L_{12}| > 2^{m/2}$. Similarly, let $L_{34} = \{(\mathbf{x}_3 \oplus \mathbf{x}_4, \mathbf{x}_3, \mathbf{x}_4) : \mathbf{x}_3 \in L_3, \mathbf{x}_4 \in L_4\}$. For each $\mathbf{v} \in L_{34}$ search for $\mathbf{v}$ in the list L_{12}. By the birthday paradox (the "two lists version") one expects a match if $|L_{12}|, |L_{34}| > 2^{m/2}$. In other words, there is a $\tilde{O}(2^{m/2})$ algorithm.

Now let us consider our problem. Suppose L_1, L_2, L_3, L_4 are lists of vectors in $\{-1,1\}^m$ of the same size chosen independently and uniformly at random. We seek $\mathbf{x}_i \in L_i$ such that $\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 = \mathbf{0}$.

To follow the same steps as above, compute a list L_{12} of sums $\mathbf{x}_1 + \mathbf{x}_2$ and a list L_{34} of sums $\mathbf{x}_3 + \mathbf{x}_4$, where, $\mathbf{x}_i \in L_i$.

$$L_{12} = \{(\mathbf{x}_1 + \mathbf{x}_2) \ : \ \mathbf{x}_1 \in L_1, \mathbf{x}_2 \in L_2\}$$

$$L_{34} = \{(-\mathbf{x}_3 - \mathbf{x}_4) \ : \ \mathbf{x}_3 \in L_3, \mathbf{x}_4 \in L_4\}.$$

Let $\mathcal{D}$ be the distribution for $\mathbf{x}_i + \mathbf{x}_{i+1}$ and note that $\mathbf{x}_i + \mathbf{x}_{i+1} \in \{-2,0,2\}^m$ for $i = 1,3$. $\mathcal{D}$ is of size 3^m. Any vector appearing in both L_{12} and L_{34} yields a solution and such a vector can be found in time linear in the lengths of L_{12} and L_{34}. Assuming $\mathcal{D}$ is uniform, the birthday paradox tells us that if your list sizes $|L_{12}|, |L_{34}| = \Theta\big(3^{m/2}\big)$, a collision is likely to happen with good probability. Such an algorithm have a running time approximately $2^{m/2\log_2 3} \approx 2^{0.792m}$.

We can also use the results from Selivanov [Sel95] to analyse this non-uniform birthday problem. The lists L_{12} and L_{34} are of vectors with entries in $\{-2,0,2\}$. The probability that an entry is 0 is $p_0 = 1/2$, and the probability of $+2$ and -2 both $p_{-2} = p_2 = 1/4$. We sort L_{12} with respect to the first coordinate and ask for the probability that there exists $\mathbf{v} \in L_{34}$ such that $\mathbf{v} \in L_{12}$.

Let p_v be the probability of a vector $\mathbf{v} \in \{-2,0,2\}^m$ being generated in a list. We get p_i by just applying the above probability distribution to each entry. Selivanov defines $v_2 = \sum_{\mathbf{v}} p_v^2$. In our case, one can see that

$$v_2 = (p_{-2}^2 + p_0^2 + p_2^2)^m = 0.375^m.$$

Selivanov treats the problem as a "coloured" problem. The vectors in L_{12} are one colour and the vectors in L_{34} are the other colour. Colours are chosen independently with probability $1/2$. The expected number of trials until the same vector $\mathbf{v}$ is chosen but coloured in both ways is,

$$\sqrt{\pi/v_2}(1 + o(1)).$$

This evaluates in our 4-list example to $\sqrt{\pi}\, 2.66^{m/2} = \sqrt{\pi}\, 2^{0.706m}$.

This immediately implies that we expect to get a match when $|L_{12}|, |L_{34}| \approx (\sqrt{\pi}/2)2^{0.7m}$. From this again we deduce that the original lists L_i should have size $2^{0.35m}$ and the algorithm complexity is $\tilde{O}(2^{0.7m})$.

Therefore, through these two approaches, it is evident that even the simplest 4-list version of the problem exhibits higher complexity than the binary case when approached using meet-in-the-middle techniques. Based on this information, we explore the following research question in this chapter.

Research Problem: How can Wagner's algorithm be adapted to solve the problem defined in Definition 6.2 over the space $\mathbb{Z}^m$, and what impact does this adaptation have on the algorithm's time and space complexity?

6.2 The new k-tree algorithm

In this section we will walk the reader through all the steps on how to solve our new variant of the generalised birthday problem using a k-sum algorithm. This includes probability calculations at each level and determining the optimal list lengths in order to achieve minimum running times for the algorithm.

The basic idea for the algorithm is simple: First find pairs $\mathbf{x}_1 \in L_1$ and $\mathbf{x}_2 \in L_2$ such that the first l_1 entries of the vector $\mathbf{x}_1 + \mathbf{x}_2$ are zero. Create a new list of such vectors $\mathbf{x}_1 + \mathbf{x}_2$. These are "partial solutions" in the sense that the problem is solved with respect to the first l_1 entries of the vector. One difficulty is that the remaining entries of the vector are in $\{-2, 0, 2\}$, but the good news is that 0 is still the most likely value in all the remaining positions. Keep on finding these "partial solutions" for different blocks of coordinates in each level of the k-tree until we end up with a zero vector of m dimension.

Our approach is heavily influenced by the work of Wagner [Wag02] and Minder-Sinclair [MS12]. We follow the same assumption as that of Wagner for initial list length. That is, we assume the initial list length is as large as needed. On the other hand, we vary the number of positions for which the problem is solved at each step, which is a strategy used by Minder and Sinclair. This problem is more complex than previous works, as at each round the

non-zero entries in the vectors come from a bigger set, and so the probability of getting zero entries changes. Therefore, we must address these challenges when analysing the new algorithm.

Notations:

From this point on, we use the following notations to name the lists in each level. The i^{th} list at the j^{th} level is denoted by $L_i^{(j)}$. In particular, we start with $L_1^{(0)}, L_2^{(0)}, \ldots, L_k^{(0)}$ in the initial level. In the next level we have $L_1^{(1)}, L_2^{(1)}, \ldots, L_{k/2}^{(1)}$, and so on down to just $L_1^{(q)}$ at the last level where $k = 2^q$. Moreover, $\mathbf{x}_i^{(j)}$ holds implicitly to represent any vector in the corresponding list $L_i^{(j)}$. The addition $(+)$ is the usual component-wise vector addition of $\{-1, 1\}^m$ vectors.

6.2.1 Overview of the Algorithm

We assume $k = 2^q$ for some integer q. Given an instance of the generalised birthday problem as outlined in Definition 6.2, the k-tree algorithm for $\{-1, 1\}^m$ vectors operates over q levels. At each level, pairs of lists (of partial solutions) are merged to form a new list (of vectors that are closer to a full solution), reducing the total number of lists by half in every round.

Let us re-name the initial k lists as $L_i^{(0)}$. In the first level, the lists $L_1^{(0)}$ and $L_2^{(0)}$ are merged to form a new list $L_1^{(1)}$, the lists $L_3^{(0)}$ and $L_4^{(0)}$ are combined into a list $L_2^{(1)}$, and the process continues for the remaining pairs. Specifically, the new list is composed of all the sums $\mathbf{x}_{2i-1}^{(0)} + \mathbf{x}_{2i}^{(0)}$ with $\mathbf{x}_{2i-1}^{(0)} \in L_{2i-1}^{(0)}$ and $\mathbf{x}_{2i}^{(0)} \in L_{2i}^{(0)}$ for $(i = 1, \ldots k/2)$ such that $\mathbf{x}_{2i-1}^{(0)} + \mathbf{x}_{2i}^{(0)}$ is zero on the first l_1 coordinates. Note that after merging, the new lists contain vectors of the form $\{-2, 0, 2\}^m$. The integer l_1 is a parameter of the algorithm that is to be determined for optimal performance. We say that the first round eliminates l_1 coordinates or equivalently make first l_1 entries zero.

To merge the two lists efficiently, let $L_1^{(0)}$ and $L_2^{(0)}$ be the input lists and suppose $|L_1^{(0)}| \leq |L_2^{(0)}|$. We impose the natural lexicographic ordering on vectors in $\mathbb{Z}^m$. Sort $L_1^{(0)}$ with respect to this ordering. This takes $\mathcal{O}\Big(|L_1^{(0)}| \log(|L_1^{(0)}|)\Big)$ operations. Then, for each element in $L_2^{(0)}$, look up in the sorted list $L_1^{(0)}$. Trying all values and searching takes $\mathcal{O}\Big(|L_2^{(0)}| \log(|L_1^{(0)}|)\Big)$ operations. So the total cost to merge the lists is bounded by $\mathcal{O}\Big(|L_2^{(0)}| \log(|L_1^{(0)}|)\Big)$ operations.

Subsequent rounds follow a similar process to the first one. In the second round, new lists $L_1^{(2)}, \ldots, L_{k/4}^{(2)}$ are formed from $L_1^{(1)}, \ldots, L_{k/2}^{(1)}$, by eliminating the next sequence of l_2

entries. This results in the vectors in the lists $L_1^{(2)}, \ldots, L_{k/4}^{(2)}$ having zeros in the first $l_1 + l_2$ coordinates, while the remaining take values from the set $\{\pm 4, \pm 2, 0\}$.

By iterating this procedure for q rounds, we obtain a final single list containing vectors that are zero on the first $\sum_{j=1}^{q} l_j$ entries. Each of these vectors is a sum of the form $\sum_{i=1}^{k} \mathbf{x}_i^{(0)}$ with $\mathbf{x}_i^{(0)} \in L_i^{(0)}$. Since the goal is to find sums that are zero on all m entries, the final list will contain sums of the desired form if the following condition holds

$$\sum_{j=1}^{q} l_j \geq m. \tag{6.1}$$

Since the input lists are random, the length of the lists at all levels are random variables. We will write m_j for the expected size of a list at level j. The distribution of m_j is determined by the initial list length m_0 and the values $l_1, \ldots, l_j$. Note that, at the last step, m_q represents the total number of solutions found by the algorithm. So our goal is to ensure that

$$m_q = |L_1^{(q)}| \geq 1. \tag{6.2}$$

The choice of l_j has a significant effect on the behaviour of the algorithm since it determines both the number of entries eliminated and the expected list length at each level. Since the running time is essentially proportional to the sum of the list lengths at each step, achieving optimal performance requires selecting l_j in such a way that the expected list length, $\mathbb{E}[m_j]$, is not too large for all $1 \leq j \leq q-1$ while satisfying the constraints (6.1) and (6.2).

Figure 6.2 illustrates the basic idea behind the algorithm for the case of $k = 8$ lists. Note that in the figure, we set $l_3 = m - l_1 - l_2$ since we are looking for a vector of dimension m.

6.2.2 Probability Calculation

In this section, we discuss how to compute the relevant probability distributions for vectors over the integers. Unlike in the binary case, when the k-list algorithm starts with vectors from $\{-1, 1\}^m$, the range of possible vector entries expands at each level of the algorithm. For instance, after the first merging step, the vectors lie in $\{-2, 0, 2\}^m$, and in the following step they expand further to $\{-4, -2, 0, 2, 4\}^m$.

To analyse the performance of the algorithm, we need to characterise the distribution of the vector entries at each level, as the success probability of zeroing out a subset of coordinates depends directly on this distribution. For example, in the second step, an entry of -2 arises uniquely from summing two -1 values, an entry of 2 from summing two $+1$ values, while a zero can result in two distinct ways: either $(-1) + 1$ or $1 + (-1)$.

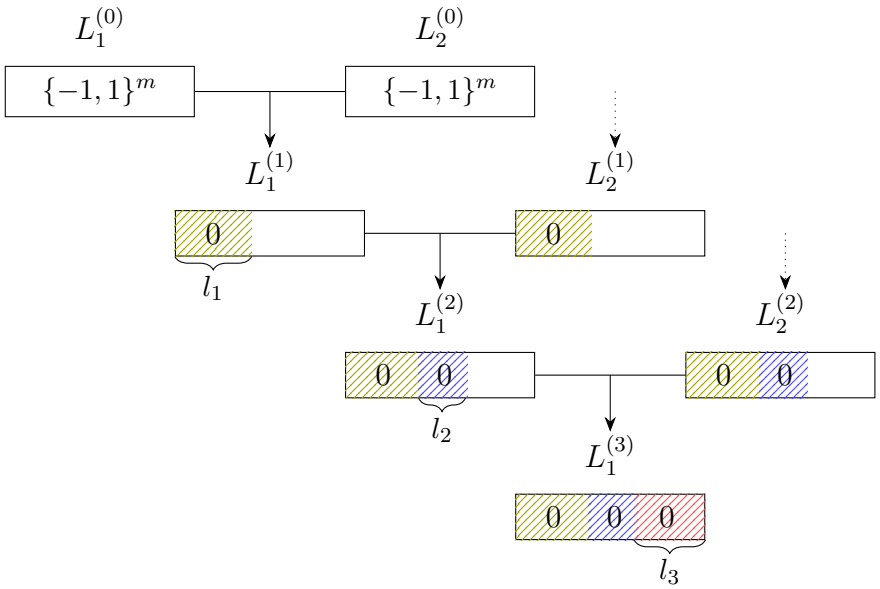


Figure 6.2: 8-list algorithm for the new generalised birthday problem

The goal of this section is to derive the general distribution of vector entries at each merging step and, in particular, to compute the probability that a given set of coordinates equals zero.

Rademacher Distribution and Random walks

A discrete probability distribution in which a random variable X takes the value $+1$ with probability 0.5 and -1 with probability 0.5 is known as the *Rademacher distribution*. A sum of independent Rademacher variables can be interpreted as a simple symmetric random walk on the integer line, where each step moves either one unit to the left or right with equal probability. In other words, adding random $\{1,-1\}$ values corresponds to a one-dimensional random walk, which is sometimes referred to as a "drunkard's walk."

In such a walk, after n steps, the maximum possible position is n and the minimum is $-n$. The distribution of possible positions at each step follows a binomial pattern, which can be derived from the properties of Pascal's triangle.

Pascal's Triangle. Pascal's triangle is a triangular array of binomial coefficients and, in our context, it directly corresponds to the probability distribution of the position in a random walk after n steps, scaled by 2^n. Consider a random walk starting at the origin (0), where each step is independently chosen as $+1$ or -1 with equal probability. Let X_n denote the random variable representing the position after n steps.

For simplicity and relevance to our analysis, we assume n is even, since otherwise the walker cannot end up at position zero. The probability of being at position $p' \in \mathbb{Z}$ (with $-n \leq p' \leq n$) after n steps is given by

$$\Pr(X_n = p') = \begin{cases} 2^{-n}\binom{n}{\frac{n+p'}{2}} & \text{if } p' \text{ is even,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.3}$$

Thus, the number of distinct, non-zero probabilities is $n+1$, corresponding to the positions $p' = -n+2i$ for $0 \leq i \leq n$. The i^{th} non-zero probability corresponds to $\Pr(X_n = -n+2i) = 2^{-n}\binom{n}{i}$.

This binomial distribution forms the foundation for understanding how vector entries evolve in our algorithm. Since we are pairing the lists and adding the vectors from these lists at each step, note that after j levels in our algorithm, we have added 2^j many vectors from the original $\{-1,1\}^m$ input lists. Therefore, at level j of the k-tree algorithm, we set $n = 2^j$ in Pascal's triangle. This is because, at each level j, the process can be viewed as having taken 2^j steps in a symmetric random walk over $\mathbb{Z}$. To illustrate, at level 1 we compute sums like $\mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)}$, while at level 2, we sum partial results: $\mathbf{x}_1^{(1)} + \mathbf{x}_2^{(1)} = (\mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)}) + (\mathbf{x}_3^{(0)} + \mathbf{x}_4^{(0)})$, and so on. Thus, by level j, each resulting vector is the sum of 2^j vectors from the initial lists.

After taking $n = 2^j$ steps in the random walk, we are particularly interested in the probability that a particular set of coordinates in the resulting vector is zero, i.e., the walk returns to the origin. According to Equation (6.3), this probability is given by $\Pr(X_n = 0) = 2^{-n}\binom{n}{n/2}$ for a 1-dimensional random walk.

However, at each step we require a block of l_j entries in the vector to be zero. Therefore, we have to raise the above probability to the power l_j to calculate the actual probability ($\Pr_j$) of getting zero in the required l_j coordinates of these vectors at each step $1 \leq j \leq q-1$. At the end of the algorithm we need to get a zero vector and hence all the l_j coordinates should add up to the dimension m of the vector. Therefore, in the final step ($j = q$), the probability must be raised to the power $l_q = m - \sum_{j=1}^{q-1} l_j$.

In summary, the probability that a random vector at a given level j containes a specific set of l_j zero entries is given by

$$\Pr{}_j = \left(2^{-n}\binom{n}{n/2}\right)^{l_j} \tag{6.4}$$

where $n = 2^j$ and l_j is a parameter to be determined.

6.2.3 Expected List Length

Once we have the probabilities of getting zero at each step, we can determine the expected list length at these steps in the k-lists algorithm. Similar to previous work in the literature, we make the following assumption about all the internal lists.

Condition 2. *All the list entries behave as independent random samples.*

Each list $L_i^{(j)}$ contains pairs of the form $(\mathbf{x}_i^{(j)}, \pi_i^{(j)})$, where:

- $\mathbf{x}_i^{(j)} \in \mathbb{Z}^m$ is a vector constructed at level j, and
- $\pi_i^{(j)} = \left(\pi_{2i-1}^{(j-1)} \parallel \pi_{2i}^{(j-1)}\right)$ with $\pi_i^{(0)} = \mathbf{x}_i^{(0)}$, is a back-pointer indicating how $\mathbf{x}_i^{(j)}$ was formed. The symbol $\parallel$ indicates the concatenation of two strings.

The initial lists are,

$$L_i^{(0)} = \left\{ \left(\mathbf{x}_i^{(0)}, \pi_i^{(0)}\right) \;\middle|\; \mathbf{x}_i^{(0)} \hookleftarrow \$ \; \{-1,1\}^m \right\},$$

where $\pi_i^{(0)} = \mathbf{x}_i^{(0)}$ is the vector itself, that used to keep track of these initial vectors. Then for $i = 1, \ldots, k/2^j$ at each level $j = 1, \ldots, \log k$, we define the list as

$$L_i^{(j)} = \left\{ \left(\mathbf{x}_i^{(j)}, \pi_i^{(j)}\right) \;\middle|\; \begin{array}{l} (\mathbf{x}_{2i-1}^{(j-1)}, -) \in L_{2i-1}^{(j-1)}, \quad (\mathbf{x}_{2i}^{(j-1)}, -) \in L_{2i}^{(j-1)}, \\ \mathbf{x}_i^{(j)} = \mathbf{x}_{2i-1}^{(j-1)} + \mathbf{x}_{2i}^{(j-1)}, \left(\mathbf{x}_i^{(j)}\right)_{l_1+\cdots+l_j} = \mathbf{0} \\ \pi_i^{(j)} = \left(\pi_{2i-1}^{(j-1)} \parallel \pi_{2i}^{(j-1)}\right) \end{array} \right\}. \tag{6.5}$$

In the definition above, we use $(\mathbf{x}_{2i-1}^{(j-1)}, -) \in L_{2i-1}^{(j-1)}$ and $(\mathbf{x}_{2i}^{(j-1)}, -) \in L_{2i}^{(j-1)}$ to indicate that we only take the *first entry* of each pair in the list, i.e., the vector itself, in order to compute the sum. The second entry (the back-pointer) is preserved in the new pair that records how this sum was formed.

This construction ensures that each vector $\mathbf{x}_i^{(j)}$ is the sum of two vectors from level $j-1$ that passed the merging condition on the specific coordinates (l_j), and that we retain sufficient information to trace back to the original inputs at level 0.

Remark 6.1. *For the remainder of this thesis, we use the notation* $\left(\mathbf{x}_i^{(j)}\right)_{l_1+\cdots+l_j} = \mathbf{0}$ *to denote that the first* $l_1 + \cdots + l_j$ *coordinates of* $\mathbf{x}_i^{(j)}$ *are zero. However, at each level* j *one only needs to consider the last* l_j *block of coordinates, since the earlier blocks are already guaranteed to be satisfied. Therefore, when there is no risk of confusion, we sometimes simply write* $\left(\mathbf{x}_i^{(j)}\right)_{l_j} = \mathbf{0}$.

We treat each $\mathbf{x}_i^{(j)}$ as an independent random vector and also, we need these vectors $\mathbf{x}_i^{(j)}$ to satisfy the special property of having zero entries in specific positions. Suppose we are starting with random lists of size m_0 and write m_j for the expected length of a list at level j of the algorithm. The distribution of m_j is determined by the values m_0 and probability $\Pr_j$ of getting zero at l_j positions. Following lemma explains how the linearity of expectation is used to derive the expected list sizes m_j.

Lemma 6.2. *Let $L_{2i-1}^{(j-1)}$ and $L_{2i}^{(j-1)}$ be two lists at level $j-1$, each of expected size m_{j-1}. Suppose that each vector in level j is formed by summing one vector from each list and retaining only those sums that satisfy the zero condition in the prescribed l_j coordinates. If $\Pr_j$ denotes the probability that the sum of a random pair satisfies this condition, then the expected size of the resulting list $L_i^{(j)}$ is*

$$m_j = \mathbb{E}\left[|L_i^{(j)}|\right] = m_{j-1}^2 \Pr{}_j.$$

Proof. At level j, each new vector $\mathbf{x}_i^{(j)}$ is obtained as the sum of two vectors $u \in L_{2i-1}^{(j-1)}$, and $v \in L_{2i}^{(j-1)}$. For each pair (u, v), define the indicator random variable

$$I_{u,v} = \begin{cases} 1, & \text{if } u+v \text{ satisfies the zero condition in the } l_j \text{ coordinates,} \\ 0, & \text{otherwise.} \end{cases}$$

Hence, the number of retained vectors in $L_i^{(j)}$ is $|L_i^{(j)}| = \sum_{(u,v)} I_{u,v}$, where the sum ranges over all pairs $(u, v) \in L_{2i-1}^{(j-1)} \times L_{2i}^{(j-1)}$.

By linearity of expectation,

$$\mathbb{E}\left[|L_i^{(j)}|\right] = \mathbb{E}\left[\sum_{(u,v)} I_{u,v}\right] = \sum_{(u,v)} \mathbb{E}[I_{u,v}] = \sum_{(u,v)} \Pr(I_{u,v} = 1).$$

Since each pair satisfies the zero condition with probability $\Pr_j$, and there are

$$|L_{2i-1}^{(j-1)}|\,|L_{2i}^{(j-1)}| = m_{j-1}^2$$

such (u, v) pairs in expectation, it follows that

$$m_j = \mathbb{E}\left[|L_i^{(j)}|\right] = m_{j-1}^2 \Pr{}_j.$$

□

For example, consider the expected list length at step 1 of the k-tree algorithm. From Equation (6.4) we have

$$\Pr{}_1 = [\Pr(X_2 = 0)]^{l_1} = \left(\frac{1}{2}\right)^{l_1}$$

when $\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}$ are chosen uniformly at random such that $\mathbf{x}_1^{(0)} \in L_1^{(0)}$ and $\mathbf{x}_2^{(0)} \in L_2^{(0)}$. We started with lists of size m_0. Then by the linearity of expectation, the expected number of $\mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)}$ in the new list is

$$m_1 = \mathbb{E}\left[|L_1^{(1)}|\right] = |L_1^{(0)}||L_2^{(0)}| \Pr{}_1 = m_0^2 \left(\frac{1}{2}\right)^{l_1}.$$

Analogously, one can compute the expected size of the list $L_2^{(1)}$ of all sums of vectors in $L_3^{(0)}$ and $L_4^{(0)}$ such that first l_1 entries are zero. The expected list size of each $L_i^{(1)}$ will be equal to the product of probability $\Pr_1$ and the size of the two merging lists $L_{2i-1}^{(0)}$ and $L_{2i}^{(0)}$ for $i = 1, \ldots, k/2$.

We keep iterating this way for q rounds. The expected list length at the j^{th} step is given by

$$m_j = m_{j-1}^2 \Pr{}_j \text{ for } j = 1, \ldots, q. \tag{6.6}$$

One can re-write this as

$$m_j = \prod_{i=1}^{j} (\Pr{}_i)^{2^{j-i}} m_0^{2^j}$$

and at the final step $j = q$,

$$m_q = \mathbb{E}\left[|L_1^{(q)}|\right] = \prod_{i=1}^{q} (\Pr{}_i)^{2^{q-i}} m_0^{2^q}.$$

After q steps, we want a final list containing at least one vector which is a zero vector or the sum of first l_j entries ($\sum_{j=1}^{q} l_j \geq m$) are zero. So our goal is to ensure that $m_q = \mathbb{E}\left[|L_1^{(q)}|\right] \geq 1$.

Algorithm 5 gives the general formulation of the k-list algorithm. The algorithm returns a set of vectors from the initial k lists that add up to zero or $\perp$ to indicate that none were found. If we start with large enough sets, the algorithm succeeds with high probability and returns a solution.

Algorithm 5 k-List Algorithm for $\{-1, 1\}^m$ vectors.

Input: $k = 2^q$, $L_i^{(0)}$ for $i = 1, \ldots, k$.
Output: $(\mathbf{x}_1^{(0)}, \cdots, \mathbf{x}_k^{(0)})$ or $\perp$

for $j = 1, \ldots, q$ **do**
  Choose l_j such that $\sum_{j=1}^{q} l_j = m$.
  for $i = 1, \ldots, k/2^j$ **do**
    $L_i^{(j)} \leftarrow \emptyset$
    Sort $L_{2i-1}^{(j-1)}$ with respect to the first coordinate, $\mathbf{x}_{2i-1}^{(j-1)}$
    for each $(\mathbf{x}_{2i}^{(j-1)}, \pi_{2i}^{(j-1)}) \in L_{2i}^{(j-1)}$ **do**
      Find all $(\mathbf{x}_{2i-1}^{(j-1)}, \pi_{2i-1}^{(j-1)}) \in L_{2i-1}^{(j-1)}$ such that $(\mathbf{x}_{2i-1}^{(j-1)} + \mathbf{x}_{2i}^{(j-1)})_{l_1+\cdots+l_j} = \mathbf{0}$.
      for all such pairs **do**
        $\mathbf{x}_i^{(j)} \leftarrow \mathbf{x}_{2i-1}^{(j-1)} + \mathbf{x}_{2i}^{(j-1)}$
        $\pi_i^{(j)} \leftarrow \left(\pi_{2i-1}^{(j-1)} \parallel \pi_{2i}^{(j-1)}\right)$
        Add $(\mathbf{x}_i^{(j)}, \pi_i^{(j)})$ to $L_i^{(j)}$
      end for
    end for
  end for
end for
if $|L_1^{(q)}| \geq 1$ **then**
  return $\pi_1^{(q)}$
else
  return $\perp$
end if

6.2.4 Running time

To derive the running time estimate for the k-lists algorithm, we carefully account for the cost of operations at each level of the list-merging process. At each internal step $j \in \{1, \ldots, q-1\}$, the algorithm maintains a collection of 2^{q-j} intermediate lists, each containing approximately m_j elements, resulting from sorting and searching elements from the previous level. We perform sorting of the lists according to lexicographic order on vectors in $\mathbb{Z}^m$, which has a cost $\mathcal{O}(m_j \log m_j)$ per list. The cost of searching for matching elements is of the same order, under the assumption that all lists at level j are of equal size m_j. Hence, the total cost to merge list pairs at level j is approximately $2^{q-j} m_j \log m_j$.

The total running time is essentially proportional to the sum of the lengths of the lists across all internal steps. Hence, we obtain the total cost for the algorithm:

$$\text{Cost} = 2^q m_0 \log(m_0) + 2^{q-1} m_1 \log(m_1) + 2^{q-2} m_2 \log(m_2) + \cdots + 2 m_{q-1} \log(m_{q-1}). \quad (6.7)$$

Note that the final list size m_q does not appear in this formula, this is because we do not need to explicitly compute the complete list of all matches, but we can stop as soon as we have found a solution as we are looking for just one solution.

For optimal performance of the algorithm we have to determine the best l_j [1] such that m_j is not too large for any $1 \leq j \leq q-1$. We also have to ensure that the constraint $m_0^k > 2^m$ holds for the existence of a solution to the k-sum problem.

The goal is to find minimum running time for the algorithm while satisfying all the above conditions. One can solve this problem as a non-linear optimisation problem or solve asymptotically for any given dimension m.

6.3 Optimised Parameters and Asymptotic Trends

As we saw in the previous section, the algorithm is specified by the parameters l_j that determine the number of bits to eliminate in each step, the initial list length m_0 and the expected list length m_j for $j = 1, \ldots, q$. Now our goal is to find an optimal choice for each l_j, m_0 and m_j that minimizes the running time while guaranteeing the existence of a solution with high probability. In this section we discuss how to solve this problem as a non-linear optimisation problem and evaluate the asymptotic behaviour for arbitrary values of m.

6.3.1 Optimisation Problem

Minder and Sinclair [MS12] reduced the problem of computing parameters to an integer program and computed the optimal values for l_j given fixed values of m_0, m, and q. In our approach, we adopt a similar framework, but instead of fixing the initial list size m_0 as they did, we allow it to be determined as part of the optimisation process. Specifically, we formulate the problem as a non-linear optimisation problem to simultaneously determine the optimal values of m_0 and l_j for given values of m and q.

This leads to the following integer program, which minimizes the cost (Equation (6.7)) subject to the given constraints.

[1] In practice, we want l_j to be integers, so good parameter choices can be obtained by using $\lceil l_j \rceil$ or $\lfloor l_j \rfloor$ appropriately. The time complexity is therefore a little bit worse, however this rounding does not affect the asymptotic complexity.

$$
\begin{aligned}
\text{minimize} \quad & \text{Cost} \\
\text{subject to} \quad & \mathbb{E}[|L_1^{(q)}|] \geq 1 \\
& m_j \geq 1 \qquad\qquad j = 1, \ldots, q-1 \\
& l_j > 0 \qquad\qquad j = 1, \ldots, q \\
& l_q = m - \sum_{j=1}^{q-1} l_j \\
& m_0 \geq 2^{m/k} \\
& m_j = m_{j-1}^2 \Pr_j \quad j = 1, \ldots, q
\end{aligned}
$$

The main condition for the algorithm to succeed is that the final list $L_1^{(q)}$ must be non-empty. To ensure this, all intermediate lists must also be non-empty, which motivates the first two constraints. Additionally, we impose the condition that at least one coordinate must be eliminated at each level, and that the total number of eliminated coordinates across all levels must equal m so that the final output is a zero vector of dimension m. The constraint $m_0 \geq 2^{m/k}$ reflects an earlier observation: this lower bound on the initial list size is a necessary condition for the success. Finally, the last constraint defines how the sizes of the intermediate lists are computed recursively.

We used the Differential Evolution method [SP97] with "best1bin" strategy to solve our problem due to its strong global search capability and robustness to non-linear, and discrete variables. It is good at searching broadly for the best solution while focusing more on promising areas to find a minimum faster.

In Table 6.1 we give an example of the result of solving this optimisation problem for $m = 100$. The cost listed in the right-hand column is a lower bound, since we have not taken into account the sorting/searching and all the low level operations. It is computed as cost $= 2^q m_0$.

k	l_1	l_2	l_3	l_4	l_5	$\log_2(m_0)$	Ω(cost)
4	44	56				42	2^{44}
8	39	24	38			37	2^{40}
16	35	21	16	27		33	2^{38}
32	32	19	15	12	22	30	2^{36}

Table 6.1: m_0, l_j and costs for different lists with $m = 100$.

The above analysis gives optimal running times for any specific instance, but it is hard to give general results. Therefore, we perform an asymptotic analysis in the next section.

6.3.2 Asymptotic analysis

In this section we make some simplifying assumptions that allow us to write down some general statements about the running time.

As we discussed earlier, the original k-tree algorithm eliminate a fixed number of $\ell = \frac{m}{q+1}$ bits in each round to keep the list length constant over all rounds, thereby balancing the work done in each round. We use a similar approach to evaluate the asymptotic trend of our algorithm.

To do this we make the simplifying assumption that, as with the original Wagner algorithm, the expected list size is constant throughout all the levels.

Condition 3. *The expected list lengths at the intermediate steps (except for the final list at level q) are equal to the initial list length m_0.*

From this we calculate the corresponding coordinates to eliminate at each level. In particular, when assumed to have equal list lengths, one can find the l_j values (number of positions to solve) at each level in terms of the probability of getting zero in that level (Pr_j) and the number of positions solved in the first step (l_1). So at the end, the initial list length and all the l_j for $j = 2, \ldots, q$ values will be in terms of l_1 which we can calculate at the last step in terms of the dimension m.

Reformulation with equal list lengths

To extend the above observations, consider an 8-list problem and refer to Figure 6.2 for clarity. A quick recap of how the algorithm works for 8 lists:

We are given the initial lists $L_1^{(0)}, \ldots, L_8^{(0)}$, and our task is to find vectors $\mathbf{x}_i^{(0)} \in L_i^{(0)}$ such that their sum equals the zero vector. A necessary condition for a solution to exist is that $|L_i^{(0)}| \geq 2^{m/8}$.

In the first level, add elements from $L_1^{(0)}$ and $L_2^{(0)}$ to generate the list $L_1^{(1)}$ such that $(\mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)})_{l_1} = \mathbf{0}$. Similarly, generate the other lists $L_2^{(1)}, \ldots, L_4^{(1)}$ by merging the initial lists pairwise. At the second level, the lists $L_1^{(2)}$ and $L_2^{(2)}$ contain vectors whose first $l_1 + l_2$ coordinates are zero. Finally, at level 3, add $\mathbf{x}_1^{(2)} \in L_1^{(2)}$ and $\mathbf{x}_2^{(2)} \in L_2^{(2)}$ to form the final list $L_1^{(3)}$, which contains vectors $\mathbf{x}_1^{(2)} + \mathbf{x}_2^{(2)}$ where the remaining coordinates satisfy

$$(\mathbf{x}_1^{(2)} + \mathbf{x}_2^{(2)})_{m-l_1-l_2} = \mathbf{0}.$$

When we assume that all the intermediate lists have the same list length as the initial lists, the complexity of the algorithm can be analysed as follows.

Using the discussion on random walks, we have $\Pr[(\mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)})_{l_1} = \mathbf{0}] = \Pr_1 = \left(\frac{1}{2}\right)^{l_1}$ when $\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)} \in \{-1, 1\}^m$ are chosen independently at random. Then, by linearity of expectation,

$$\mathbb{E}\left[|L_1^{(1)}|\right] = |L_1^{(0)}||L_2^{(0)}| \Pr_1 = \frac{m_0^2}{2^{l_1}}.$$

Assuming $\mathbb{E}\left[|L_1^{(1)}|\right] \approx m_0$, we simplify to get $m_0 = 2^{l_1}$.

At level 2, we consider vectors from $L_1^{(1)}$ and $L_2^{(1)}$ that sum to zero in the next l_2 coordinates. Using the same reasoning, we write the expected size of $L_1^{(2)}$ as

$$\mathbb{E}\left[|L_1^{(2)}|\right] = |L_1^{(1)}||L_2^{(1)}| \Pr_2 = m_0^2 \left(\frac{6}{2^4}\right)^{l_2} \approx m_0.$$

Substituting $m_0 = 2^{l_1}$ gives $l_2 = \frac{-l_1}{\log_2(3/8)} \approx 0.707 l_1$.

Finally, at the last level, we want the expected size of $L_1^{(3)}$ to be at least 1

$$\mathbb{E}\left[|L_1^{(3)}|\right] = |L_1^{(2)}||L_2^{(2)}| \Pr_3 = m_0^2 \left(\frac{70}{2^8}\right)^{m-l_1-l_2} \geq 1.$$

By substituting the values for m_0 and l_2 derived in the previous steps, we can solve for l_1 in terms of the dimension m of the vectors. This gives the number of entries to eliminate at the first level. By back-substitution, we obtain the required initial list lengths (m_0) needed to find a solution, as well as the number of coordinates to eliminate at the subsequent levels (l_j).

We observed that if we set $l_1 = 0.37m$, we can expect to find a solution to the $8-$sum problem with non-trivial probability. Under this assumption, the size of all the lists will be approximately $m_0 = 2^{0.37m}$, and hence the algorithm can be implemented with time and space complexity $\mathcal{O}(2^{0.37m})$, up to logarithmic factors.

This idea can be extended to any $k = 2^q$ number of lists. When k is a power of 2, we obtain a binary tree of depth $\log_2 k = q$. At each internal node, we maintain the assumption that the expected list length equals the initial list length m_0. This allows us to express the values of m_0 and l_j in terms of l_1. More precisely, we have

$$m_0 = 2^{l_1} \quad \text{and} \quad l_j = \frac{-l_1}{\log_2(\Pr_j)} \quad \text{for } j = 2, \ldots, q-1, \tag{6.8}$$

where $\Pr_j$ is the probability of landing at zero after $n = 2^j$ steps in a 1-dimensional random walk (i.e., Equation (6.3) with $p' = 0$).

In the final level q, we set $l_q = m - \sum_{j=1}^{q-1} l_j$ to ensure that we obtain the zero vector of dimension m. If we want to find a single solution to the k-sum problem, we set the expected list length at the final level to be 1

$$\mathbb{E}\left[|L_1^{(q)}|\right] = m_0^2 \left(\Pr{}_q\right)^{m-\sum_{j=1}^{q-1} l_j} = 1.$$

Substituting the expressions for m_0 and l_j and taking the logarithm, we solve for l_1 as follows

$$m - \sum_{j=1}^{q-1} l_j = \frac{-2l_1}{\log_2(\Pr_q)}.$$

Since $\sum_{j=1}^{q-1} l_j = l_1 + \sum_{j=2}^{q-1} \frac{-l_1}{\log_2 \Pr_j}$, we obtain

$$l_1 = \frac{m}{1 - \sum_{j=2}^{q-1} \frac{1}{\log_2 \Pr_j} - \frac{2}{\log_2 \Pr_q}}. \tag{6.9}$$

This framework enables us to solve the k-sum problem for any given dimension m with both time and space complexity bounded by $\mathcal{O}(k \cdot m_0 \log m_0)$. The algorithm operates using intermediate lists of size $\mathcal{O}(m_0)$ at each level of the merging process, where $m_0 = 2^{l_1}$ and l_1 is determined as a function of the dimension m and the collision probabilities $\Pr_j$ associated with the merging steps.

The simplicity of this variant, particularly the assumption that all intermediate lists maintain equal size makes the algorithm straightforward to analyse. This allows us to derive clean asymptotic bounds and make general statements about the algorithm's performance in terms of how it scales with k and m.

Theorem 6.2. *Under the assumption that the input lists are independently and uniformly distributed, and that all intermediate lists produced during the algorithm have equal size, the k-sum problem (as defined in Definition 6.2) can be solved by the algorithm with an expected running time bounded asymptotically by $\mathcal{O}(k \cdot m_0 \log m_0)$ and memory usage bounded by $\mathcal{O}(k \cdot m_0)$, where $m_0 = 2^{l_1}$ and the parameter l_1 is a function of the vector dimension m and the collision probabilities $\Pr_j$ associated with the merging steps, as specified in Equation (6.9).*

Proof. The total running time is proportional to the sum of the lengths of the lists across all internal steps. At any internal merging step j, the algorithm maintains 2^{q-j} lists, and under the Condition 3, these lists are all of the equal length m_0 throughout the process.

For each merging step, the most computationally intensive operations are sorting and searching, both of which require $\mathcal{O}(m_0 \log m_0)$ time per list. Hence, the total computational

cost incurred during step j is $\mathcal{O}(2^{q-j} \cdot m_0 \log m_0)$. Summing over all q merging steps, the overall running time is given by:

$$\text{Cost} = \sum_{j=1}^{q} 2^{q-j} \cdot m_0 \log m_0 = m_0 \log m_0 \sum_{j=1}^{q} 2^{q-j} = m_0 \log m_0 \cdot (2^q - 1).$$

Since $k = 2^q$ represents the total number of original input lists, this simplifies to:

$$\text{Running time} = \mathcal{O}(k \cdot m_0 \log m_0).$$

Regarding memory consumption, the algorithm stores k lists of length m_0 at the initial step, and the number of lists are halved at each intermediate steps. Thus, the space complexity is $\mathcal{O}(k \cdot m_0)$, neglecting lower-order and logarithmic factors. □

Experimental results:

Table 6.2 presents a comparison of how the number of coordinates eliminated at each merging level (denoted by l_j) changes as a function of the number of input lists $k = 2^q$. The values are expressed relative to an arbitrary vector dimension m, allowing us to observe how the elimination schedule adapts as the tree depth increases with larger k.

k	l_1	l_2	l_3	l_4	l_5	l_6	l_7
4	$0.4144m$	$0.5856m$					
8	$0.3603m$	$0.2545m$	$0.3852m$				
16	$0.3233m$	$0.2285m$	$0.1728m$	$0.2754m$			
32	$0.2966m$	$0.2096m$	$0.1585m$	$0.1263m$	$0.2090m$		
64	$0.2762m$	$0.1952m$	$0.1477m$	$0.1176m$	$0.0974m$	$0.1659m$	
128	$0.2603m$	$0.1839m$	$0.1391m$	$0.1108m$	$0.0917m$	$0.0781m$	$0.1361m$

Table 6.2: Comparison of number of lists ($k = 2^q$) and l_j values for $j = 1, \ldots, q$

In Figure 6.3, we illustrate how the running time of the algorithm varies with the number of lists $k = 2^q$, across different vector dimensions m. A common characteristic of Wagner-type algorithms is a "U"- shaped cost curve: as k increases, the algorithm initially benefits from smaller required list sizes, which reduce the cost of intermediate merging steps. However, these gains are eventually outweighed by the exponential cost introduced by the 2^q factor in the total running time, where $k = 2^q$. This trade-off is particularly evident in the cases of $m = 100$ and $m = 200$, where the minimum running time is achieved at $k = 2^9$ and $k = 2^{14}$ respectively.

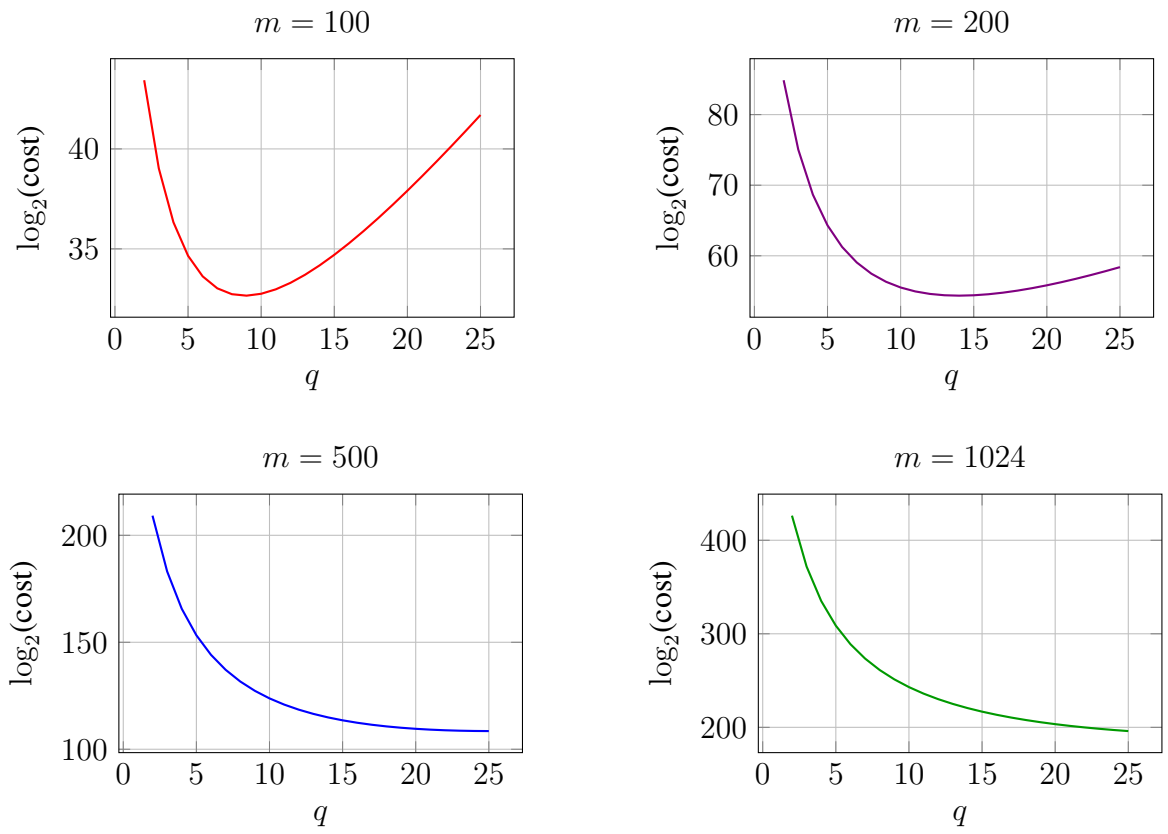


Figure 6.3: Variation of cost with the number of lists for $m = 100$, 200, 500, and 1024

For larger dimensions such as $m = 500$ and $m = 1024$, the figure does not extend far enough to clearly observe the "U" shape. However, the trend indicates that increasing the number of lists beyond $k = 2^{20}$ provides diminishing returns in terms of performance improvement. Therefore, it is important to identify the optimal k that minimizes the overall running time.

Remark 6.2. *To compare our results with Wagner's original algorithm for binary vectors, consider the version that uses 4 lists. Recall that for binary vectors, Wagner's algorithm with initial list sizes of* $m_0 = 2^{m/3}$ *runs in time approximately* $\mathcal{O}(2^{0.3m})$*. Now, looking at our results in Table 6.2 and Figure 6.3, we see that for vectors over* $\{-1, 1\}^m$*, the* $4-$*list algorithm has a higher complexity, around* $2^{0.4m}$*.*

The above Remark 6.2 provides further support for our initial claim that the generalised birthday problem with ± 1 vectors is more difficult than the version with binary vectors. Additionally, the naïve algorithm for $\{-1, 1\}^m$ vectors had a running time of roughly $\mathcal{O}(2^{0.792m})$. Using the $4-$tree algorithm, we were able to significantly reduce this to

around $\mathcal{O}(2^{0.414m})$ ignoring logarithmic factors, demonstrating a substantial improvement in efficiency.

6.4 Summary & Future work

In this chapter, we introduced a new variant of the generalised birthday problem where the input vectors are drawn from the set $\{-1, 1\}^m$, rather than the traditional binary vectors $\{0, 1\}^m$. We analysed the complexity of solving this problem using a Wagner-like k-tree algorithm, which is a structured approach inspired by Wagner's original method for binary inputs.

We initially observed that this $\{-1, 1\}$-valued version of the problem is inherently more difficult to solve than its binary counterpart. This was illustrated through a meet-in-the-middle argument in the simplest case involving 4 lists, where the increased complexity becomes evident.

To address this challenge, we proposed a k-tree algorithm, which achieves significantly lower computational complexity compared to conventional meet-in-the-middle strategies. We observed a notable reduction in running time when using these structured algorithms.

We provided detailed guidance on how to construct the k-list algorithm specifically tailored to the $\{-1, 1\}^m$ generalised birthday problem. A theoretical analysis of the proposed algorithms was presented, outlining their expected time complexity under various scenarios and assumptions.

The computational cost of the algorithm was analysed from two perspectives: first, as a non-linear optimisation problem, allowing for precise numerical evaluation; and second, through asymptotic approximations, offering insight into the algorithm's scaling behaviour for large input sizes. In both approaches, we employed heuristic assumptions to better estimate the algorithm's performance across a range of parameter settings.

An actual implementation of the full algorithm and checking the validity of the theoretical estimates would be an interesting future work.

As a direct application of our work, we apply the above algorithm to solve a new variant of the Inhomogeneous Short Integer Solution (ISIS) in the next chapter.

Chapter 7

Application to rOM-ISIS

Contents

The motivation for our study of the new k-sum problem and the corresponding k-tree algorithms developed in Chapter 6 stems from a recent cryptographic assumption known as the *randomised one-more ISIS* (rOM-ISIS) assumption. This assumption was introduced by Baldimtsi, Cheng, Goyal, and Yadav [BCGY24] as a more robust alternative to the previously established *one-more ISIS* (OM-ISIS) assumption, which was proposed by Agrawal, Kirshanova, Stehlé, and Yadav [AKSY22].

The purpose of this chapter is twofold. First, we provide a detailed comparison between the OM-ISIS and rOM-ISIS assumptions, highlighting both their structural differences and their implications in terms of cryptographic hardness. Second, we propose a concrete cryptanalytic strategy that still needs more than polynomially-many oracle queries targeting the rOM-ISIS assumption. This strategy leverages the new k-tree algorithm introduced in Chapter 6, which is specifically designed to solve the structured k-sum problem over $\{\pm 1\}^m$ vectors efficiently.

7.1 Background and Motivation

Over the years, several variants of the ISIS assumption (Definition 2.15) have been proposed. In 2022, Agrawal et al. [AKSY22] introduced a notable new variant called the *one-more ISIS* (OM-ISIS) assumption. This assumption was instrumental in proving the security of a lattice-based blind signature scheme presented within the same work. Informally, the OM-ISIS assumption states that for any polynomially bounded number ℓ, it is computationally hard to forge $\ell + 1$ valid solutions even when given oracle access up to ℓ inversions of arbitrarily chosen syndromes. We present the formal definition of the OM-ISIS assumption below:

Definition 7.1. *(One-more ISIS) Let p, n, m, β, ζ be functions of security parameter λ. Consider the following experiment:*

1. *A challenger uniformly samples a matrix $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$ and sends $\mathbf{A}$ to the adversary $\mathcal{A}$.*

2. *$\mathcal{A}$ performs two types of queries in any order.*

   Syndrome queries. *$\mathcal{A}$ requests a challenge vector, to which the challenger replies with a uniformly sampled vector $\mathbf{t} \leftarrow\!\!\$\ \mathbb{Z}_p^n$. Denote the set of received vectors by S.*

   Preimage queries. *$\mathcal{A}$ queries any vector $\mathbf{t}' \in \mathbb{Z}_p^n$. The challenger will return a short vector $\mathbf{y}' \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{y}' \equiv \mathbf{t}' \pmod p$ and $\|\mathbf{y}'\| \leq \zeta\sqrt{m}$. Let ℓ denote the number of preimage queries.*

3. *Finally, $\mathcal{A}$ outputs $\ell + 1$ distinct pairs of the form $\{(\mathbf{y}_j, \mathbf{t}_j)\}_{j \in [\ell+1]}$. $\mathcal{A}$ wins if*

$$\forall j \in [\ell + 1],\ \ \mathbf{A}\mathbf{y}_j \equiv \mathbf{t}_j \pmod p,\ \|\mathbf{y}_j\| \leq \beta,\ \textit{and}\ \mathbf{t}_j \in S.$$

The OM-ISIS assumption states that for every PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins is $\mathsf{negl}(\lambda)$.

The OM-ISIS assumption states that an efficient adversary cannot produce reasonably short preimages for $\ell + 1$ randomly sampled target vectors while being allowed to query a preimage sampling oracle on at most ℓ targets of its choosing. The hardness of this problem critically depends on several parameters, notably β, which bounds the norm of the preimage vectors $\mathbf{y}_i$ that the adversary is required to produce, as well as the dimensions m and n of the input matrix $\mathbf{A}$.

The authors argue that this new assumption is closely related to the well-studied k-ISIS problem [BF11], which, in certain special cases, can be reduced to the classic SIS problem. However, unlike the k-ISIS problem, they have not been able to establish a reduction from any widely accepted cryptographic assumption to the OM-ISIS assumption. Instead, they propose two initial cryptanalytic strategies targeting OM-ISIS: a combinatorial attack and a lattice-based attack, serving as preliminary efforts to evaluate its security.

All the cryptanalysis efforts against the OM-ISIS assumption leverage the adversary's ability to submit arbitrary target vectors as preimage queries. This flexibility allows the attacker to request preimages of specially chosen vectors, such as short vectors or the zero vector, and use these responses to construct an approximate trapdoor. However, the quality of this approximate trapdoor is significantly inferior to that of the actual trapdoor. Consequently, such a trapdoor would be ineffective in producing the required $\ell + 1$ preimage vectors if these vectors must be as short as the preimages initially obtained from the oracle.

For the proposed algorithms, the authors report the following running times for solving the OM-ISIS problem:

- Combinatorial algorithm: Using $Q \geq np$ preimage queries, the problem can be solved in time Q with $\beta = \Theta\left(\sqrt{1 + \frac{n \log p}{\log(Q/n^2)}} \cdot \sqrt{m}\zeta\right)$
- Lattice-based algorithm: Using $\mathcal{O}(m^2)$ preimage queries, the problem can be solved in polynomial time with $\beta = \Theta(m\zeta)$

where ζ is a width parameter of the preimage queries.

In [BCGY24], it is argued that granting the adversary unrestricted access to the preimage query oracle is too strong and renders the OM-ISIS assumption vulnerable to arbitrary linear combination attacks. To illustrate this point, the authors present a simple attack that efficiently finds short preimage vectors without relying on the construction of a lattice trapdoor.

Suppose the attacker makes two preimage queries: one on the zero vector $\mathbf{t} = \mathbf{0}$ and another on any non-zero target vector $\mathbf{t} \in S$. Let $\mathbf{y}_1$ and $\mathbf{y}_2$ be the respective preimage vectors, so that

$$\mathbf{A}\mathbf{y}_1 = \mathbf{0} \quad \text{and} \quad \mathbf{A}\mathbf{y}_2 = \mathbf{t}.$$

Given $\mathbf{y}_1$ and $\mathbf{y}_2$, the attacker can construct three distinct vectors,

$$\mathbf{y}_2, \quad \mathbf{y}_2 - \mathbf{y}_1, \quad \text{and} \quad \mathbf{y}_2 + \mathbf{y}_1,$$

all of which serve as valid preimages for the same target vector $\mathbf{t} \in S$.

This, however, does not qualify as a practical attack on the two-round blind signature scheme presented in [AKSY22], since that scheme requires an adversary to generate preimages for $\ell + 1$ *distinct* challenge vectors. Nevertheless, the existence of such a simple attack against the underlying computational assumption raises concerns about the scheme's security. To address these issues, Baldimtsi et al. [BCGY24] proposed the *randomised one-more ISIS* (rOM-ISIS) assumption, specifically designed to thwart these types of attacks.

7.1.1 Randomised OM-ISIS assumption

The randomised one-more ISIS (rOM-ISIS) assumption was introduced to address the limitations inherent in the OM-ISIS assumption. The new assumption has two primary objectives:

(a) To protect the system against adversaries that can request multiple preimage queries on the *same* target vector,

(b) To enable re-randomisation of the target vectors before responding to each preimage query, thereby enhancing the assumption's robustness.

To realise these goals, Baldimtsi et al. [BCGY24] modify the OM-ISIS assumption in light of the practical cryptanalysis attacks demonstrated by Agrawal et al. [AKSY22], which exploit the attacker's ability to obtain preimages for arbitrary target vectors of its choice. Specifically, the challenger in the rOM-ISIS setting independently *re-randomises* each target vector before computing its preimage. This re-randomisation removes the attacker's advantage of obtaining multiple distinct short preimages for the same target vector (or any chosen target vector more generally).

We now give the formal definition of the rOM-ISIS assumption as follows:

Definition 7.2. *(randomised OM-ISIS) Let p, n, m, β, ζ be functions of security parameter λ. Consider the following experiment:*

1. *The challenger uniformly samples two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_p^{n \times m}$, and sends $\mathbf{A}, \mathbf{B}$ to adversary $\mathcal{A}$.*

2. *$\mathcal{A}$ adaptively makes queries of the following types to the challenger, in any order:*

   Syndrome queries. *$\mathcal{A}$ requests for a challenge vector, to which the challenger replies with a uniformly sampled vector $\mathbf{t} \leftarrow\!\!\$\ \mathbb{Z}_p^n$. We denote the set of received vectors by S.*

> ***Preimage queries.*** *$\mathcal{A}$ queries a vector $\hat{\mathbf{t}} \in \mathbb{Z}_p^n$, to which the challenger replies with a short vector $\hat{\mathbf{x}} \in \mathbb{Z}^m$ and a vector $\hat{\mathbf{y}} \in \{\pm 1\}^m$ such that $\mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\hat{\mathbf{y}} = \hat{\mathbf{t}}$ and $\|\hat{\mathbf{x}}\| \leq \zeta\sqrt{m}$. Let ℓ denote the total number of preimage queries.*

3. *Finally, $\mathcal{A}$ outputs $\ell + 1$ distinct tuples of the form $\{(\mathbf{x}_j, \mathbf{y}_j, \mathbf{t}_j)\}_{j\in[\ell+1]}$. $\mathcal{A}$ wins if*

$$\forall j \in [\ell+1], \quad \mathbf{A}\mathbf{x}_j + \mathbf{B}\mathbf{y}_j = \mathbf{t}_j, \quad \|\mathbf{x}_j\| \leq \beta, \quad \mathbf{y}_j \in \{\pm 1\}^m, \quad \textit{and } \mathbf{t}_j \in S.$$

The rOM-$\mathsf{ISIS}_{p,n,m,\beta,\zeta}$ assumption states that for every PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ wins is $\mathsf{negl}(\lambda)$.

According to this assumption, an adversary is no longer granted preimage access to arbitrary target vectors $\hat{\mathbf{t}}$ of its own choosing. Instead, the challenger transforms each adversarially chosen target vector $\hat{\mathbf{t}}$ into a new, "re-randomised" vector $\mathbf{t}'$ before generating the preimage: $\mathbf{t}' = \hat{\mathbf{t}} - \mathbf{B}\hat{\mathbf{y}}$, where $\hat{\mathbf{y}}$ is a randomly selected vector whose entries are independently sampled from $\{\pm 1\}$, and $\mathbf{B}$ is a public matrix determined by the challenge parameters. In effect, the preimage query phase now returns preimages for a randomised variant of the adversary's target.

Baldimtsi et al. [BCGY24] argue that this re-randomisation mechanism grants the challenger more flexibility and control over the preimage query phase. Importantly, enforcing the constraint that the vector $\hat{\mathbf{y}}$ must lie in the set of ± 1 vectors is key to preventing algebraic attacks that have been shown effective against the earlier OM-ISIS formulation.

In their work, Baldimtsi et al. [BCGY24] also analyse the resilience of the rOM-ISIS assumption against known attack strategies. They specifically revisit the cryptanalytic methods proposed by Agrawal et al. [AKSY22] including combinatorial and lattice-based approaches which were used to challenge the OM-ISIS assumption. The authors contend that, in the rOM-ISIS setting, these attacks are no longer directly applicable or become significantly less effective. The randomised nature of the target vectors effectively disrupts the attacker's ability to exploit repeated structure or to build approximate trapdoors. To support this claim, they present adaptations of both a combinatorial attack and a lattice attack, highlighting how the re-randomisation step and the ± 1 restriction limit the attack surface. We briefly discuss the essence of these attacks below (See Section 6 in [BCGY24] for further details).

Lattice Attack

The adversary begins by making Q preimage queries on the zero vector, i.e., $\mathbf{t}_i = \mathbf{0}$, and receives a set of pairs $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i\in[Q]}$ satisfying the equation $\mathbf{A}\mathbf{x}_i + \mathbf{B}\mathbf{y}_i = \mathbf{0}$, where each $\mathbf{x}_i$ is a short vector and $\mathbf{y}_i \in \{\pm 1\}^m$. The goal of the attacker is to exploit the linear structure in these vectors to create an approximate trapdoor.

If the adversary can find coefficients $a_1, a_2, \ldots, a_Q \in \mathbb{Z}$ such that $\mathbf{B}\sum_{i\in[Q]} a_i\mathbf{y}_i = \mathbf{0}$, then the corresponding linear combination $\alpha := \sum_{i\in[Q]} a_i\mathbf{x}_i$ satisfies $\mathbf{A}\alpha = \mathbf{0}$, and therefore, α is a vector in the lattice $\Lambda_p^\perp(\mathbf{A})$. If the coefficients a_i are all bounded in absolute value by a known parameter γ, then α has a norm bounded above by $\gamma\sqrt{Qm}\zeta$ with all but negligible probability. This allows the adversary to compute a short basis E for $\Lambda_p^\perp(\mathbf{A})$, such that the corresponding Gram-Schmidt basis has norm bounded from above.

Once this basis is available, the attacker proceeds to solve the rOM-ISIS problem. Given a challenge vector $\mathbf{t} \in S$, compute $(\mathbf{u}, \mathbf{y})$ such that $\mathbf{y} \in \{-1, 1\}^m$ and $\mathbf{u}$ is not necessarily short and satisfy the rOM-ISIS condition $\mathbf{A}\mathbf{u} + \mathbf{B}\mathbf{y} = \mathbf{t}$. Now the adversary runs Babai's nearest plane algorithm using the lattice basis E and the vector $\mathbf{u}$ to find a lattice vector $\mathbf{v} \in \Lambda_p^\perp(\mathbf{A})$. Then $\mathbf{u} - \mathbf{v}$ is a short vector and therefore, the tuple $(\mathbf{u} - \mathbf{v}, \mathbf{y}, \mathbf{t})$ is a valid solution to the rOM-ISIS challenge for the target $\mathbf{t}$.

The main observation is that the trapdoor obtained through this method is of lower quality compared to the one possessed by the challenger. The authors argue that it is computationally hard for an attacker to efficiently construct a good basis for the lattice $\Lambda_p^\perp(\mathbf{A})$. Consequently, the key challenge lies in refining or enhancing this degraded trapdoor to enable the sampling of sufficiently short preimage vectors.

Combinatorial attack

Let $\{\mathbf{y}'_j\}_{j\in[Q]}$ be a set of Q randomly chosen vectors from $\{-1, 1\}^m$. The adversary attempts to exploit the structure of the randomised one-more ISIS (rOM-ISIS) setting by querying the preimage oracle on a specially crafted set of target vectors. In particular, the adversary forms the set

$$A = \left\{ a_i\mathbf{e}_i + \mathbf{B}\left(\mathbf{y}'_j - n^{-1}\mathbf{1}\right) \;\middle|\; \forall i \in [n],\ \forall j \in [Q],\ a_i \in \mathbb{Z}_p \right\},$$

where $\mathbf{e}_i$ denotes the i-th standard basis (unit) vector in $\mathbb{Z}_p^n$, and n^{-1} denotes the multiplicative inverse of n modulo p.

The idea is that by summing at most n vectors of the form $a_i\mathbf{e}_i$, the attacker can construct any desired target vector $\mathbf{t} \in \mathbb{Z}_p^n$, since the standard basis vectors span the space. Suppose the oracle returns preimages $(\mathbf{x}_{i,j}, \mathbf{y}_{i,j})$ for each query in the set A. That is, for each pair

(i, j), we have

$$\mathbf{A}\mathbf{x}_{i,j} + \mathbf{B}\mathbf{y}_{i,j} = a_i\mathbf{e}_i + \mathbf{B}(\mathbf{y}'_j - n^{-1}\mathbf{1}).$$

Now, assume we are lucky enough that for each $i \in [n]$, there exists an index $j \in [Q]$ such that $\mathbf{y}_{i,j} = \mathbf{y}'_j$. Define $\mathcal{I}$ to be the set of all such matching pairs (i, j), and let J be the set of corresponding indices j.

Summing over all $(i, j) \in \mathcal{I}$, we obtain

$$\mathbf{A}\sum_{(i,j)\in\mathcal{I}} \mathbf{x}_{i,j} + \mathbf{B}\sum_{(i,j)\in\mathcal{I}} \mathbf{y}_{i,j} = \sum_{i\in[n]} a_i\mathbf{e}_i + \mathbf{B}\sum_{j\in J}(\mathbf{y}'_j - n^{-1}\mathbf{1}).$$

Since $\mathbf{y}_{i,j} = \mathbf{y}'_j$ for $(i, j) \in \mathcal{I}$ and $j \in J$, and $\sum_{i=1}^{n} a_i\mathbf{e}_i = \mathbf{t}$ under our assumption, this simplifies to

$$\mathbf{A}\sum_{(i,j)\in\mathcal{I}} \mathbf{x}_{i,j} + \mathbf{B}\cdot\mathbf{1} = \mathbf{t},$$

leading to a valid solution to the rOM-ISIS challenge. Hence, the tuple $\left(\sum_{(i,j)\in\mathcal{I}} \mathbf{x}_{i,j},\ \mathbf{1},\ \mathbf{t}\right)$ is a valid solution to the rOM-ISIS problem instance for the challenge vector $\mathbf{t}$, where the norm of $\sum_{(i,j)\in\mathcal{I}} \mathbf{x}_{i,j}$ is bounded by $\beta = \zeta\sqrt{nm}$ with non-negligible probability.

However, as the authors emphasise, the success of this attack relies on finding such a set $\mathcal{I}$ (and J) with matching vectors $\mathbf{y}_{i,j} = \mathbf{y}'_j$. The probability of such matches occurring goes to zero when $Q = o(2^m)$. Hence, this attack is computationally infeasible for any PPT adversary making polynomially many queries.

In conclusion, this combinatorial attack would definitely require more than $2^{m/2}$ preimage queries to have a meaningful chance of success, rendering it impractical in the standard security setting.

Apart from these preliminary cryptanalytic attempts, Baldimtsi et al. did not pursue further security analysis of the rOM-ISIS assumption and explicitly leave it as an open problem. This gap in the analysis motivated us to explore potential paths for cryptanalysis using well-known combinatorial algorithms.

7.1.2 Motivation & Research Problem

There exists extensive literature on the ISIS problem and various combinatorial algorithms and techniques that can be employed to tackle it, including approaches based on tree-like structures. Bai et al. [BGLS19] survey several combinatorial algorithms originally developed for the knapsack and subset-sum problems such as those by Schroeppel and Shamir [SS81], Wagner [Wag02], Howgrave, Graham and Joux [HGJ10], and Becker, Coron, and Joux [BCJ11] which have been adapted to the ISIS setting.

The fundamental idea underlying all these algorithms is to recursively reduce the problem into smaller, simpler sub-problems using a binary tree-like framework. Partial solutions obtained are then combined or "merged" at each level to reconstruct a solution to the original problem. Motivated by this approach, we believe that designing a tree-like algorithm offers a promising direction for addressing the new version of the ISIS problem.

Our approach to the problem is to identify a set of vectors $\mathbf{y}$ whose sum is zero, and then manipulate the corresponding "short" vectors $\mathbf{x}$ satisfying $\mathbf{Ax} = \mathbf{0}$ in order to construct the additional components needed for a solution. The main challenge lies in finding a set of uniformly random vectors from $\{\pm 1\}^m$ that sum to the zero vector. We observed that this task is closely related to the *generalised birthday problem*, where the goal is to find a set of binary vectors that sum to zero, except in our case, we replace binary vectors with vectors in $\{-1, 1\}^m$.

This connection inspired us to analyse the k-tree algorithm in the context of $\{-1, 1\}^m$ vectors, which led to the results presented in Chapter 6. Once we have an efficient algorithm to find a set of $\mathbf{y}$ vectors summing to zero, the remaining task reduces to identifying $\mathbf{x}$ vectors that are sufficiently short and yields a valid solution to the rOM-ISIS problem under the required norm constraints.

Both Definition 7.1 and Definition 7.2 require the challenger to return sample vectors whose norms are bounded by $\zeta\sqrt{m}$, while the adversary must produce vectors with norm at most β. We will demonstrate that the larger the gap between these two bounds, the more effective our cryptanalytic techniques become. In particular, our methods highlight the security risk of selecting β significantly larger than $\zeta\sqrt{m}$.

Remark 7.1. *There is a subtle technical point in these definitions that must be clarified for the purpose of our attacks. The preimage queries are answered using standard lattice trapdoor sampling techniques (Section 2.4.4). This means, each vector $\hat{\mathbf{x}}$ is generated using the* SamplePre *algorithm with parameter ζ. It then follows from Lemma 2.2 that, with overwhelming probability, the norm satisfies $\|\hat{\mathbf{x}}\| \leq \zeta\sqrt{m}$.*

However, for the analysis of our attacks later in the thesis, we adopt a stronger evaluation criterion: namely, that $\hat{\mathbf{x}}$ is not merely bounded in norm, but is sampled from a discrete Gaussian distribution with parameter ζ, with squared expected norm as given in Lemma 2.1. This refinement better captures the statistical properties relevant to our cryptanalysis.

7.2 New Algorithm for the rOM-ISIS Problem

We propose a new combinatorial attack strategy against the rOM-ISIS assumption, based on Wagner's k-tree algorithm. Our analysis suggests that the security of the assumption depends critically on the choice of underlying parameters. In their work, Baldimtsi et al. [BCGY24] adopt the parameter set used in Falcon-512 [FHK$^+$18] and the scheme by Agrawal et al. [AKSY22]. Specifically, they set the modulus $p = 7349$, the dimension $n = 512$, and the lattice dimension $m = 1024$. They also choose $\zeta = \omega\sqrt{n \log q \log m}$, which ensures that $\zeta \geq \eta(\epsilon)$. In this section, we analyse the impact of these parameters on the effectiveness of our attack.

At a high level, the attack strategy is as follows. The attacker makes many preimage queries on the zero vector (i.e. $\mathbf{t} = \mathbf{0} \in \mathbb{Z}_p^n$) to get many pairs $(\mathbf{x}_i, \mathbf{y}_i)$ such that $\mathbf{A}\mathbf{x}_i + \mathbf{B}\mathbf{y}_i = \mathbf{0}$, $\mathbf{y}_i \in \{-1, 1\}^m$ and $\|\mathbf{x}_i\| \leq \zeta\sqrt{m}$. Our variant of Wagner's algorithm produces a set $\mathcal{I}$ of size k such that

$$\sum_{i \in \mathcal{I}} \mathbf{y}_i = \mathbf{0}.$$

Then

$$\mathbf{A}\left(\sum_{i \in \mathcal{I}} \mathbf{x}_i\right) = \mathbf{0}.$$

Let $\mathbf{x} = \sum_{i \in \mathcal{I}} \mathbf{x}_i$. Note that each $\mathbf{x}_i$ is produced using the SamplePre algorithm for lattice parameters n, m, p, with $m \geq \mathcal{O}(n \log p)$. Therefore, each $\mathbf{x}_i$ is a solution to $\mathbf{A}\mathbf{x}_i = -\mathbf{B}\mathbf{y}_i$ for some randomly selected $\mathbf{y}_i \in \{\pm 1\}^m$, and $\|\mathbf{x}_i\| \leq \zeta\sqrt{m}$ with overwhelming probability (see Section 2.4.4).

According to the framework of trapdoor functions with preimage sampling [GPV08], the conditional minimum entropy of $\mathbf{x}_i$ is $H_\infty(\mathbf{x}_i) \geq \omega(\log m)$. Recall that the min-entropy of a discrete random variable X is defined as

$$H_\infty(X) := -\log_2(\max_x \Pr[X = x]).$$

For a discrete Gaussian $\mathbf{x}_i \leftarrow\!\!\$\ \mathcal{D}_{\mathcal{L},\zeta}$, the probability of any fixed vector $\mathbf{v} \in \mathcal{L}$ is

$$\Pr[\mathbf{x}_i = \mathbf{v}] = \frac{e^{-\pi\|\mathbf{v}\|^2/\zeta^2}}{\sum_{\mathbf{z} \in \mathcal{L}} e^{-\pi\|\mathbf{z}\|^2/\zeta^2}}$$

The numerator, $e^{-\pi\|\mathbf{v}\|^2/\zeta^2}$, decreases as $\|\mathbf{v}\|$ increases. Therefore the maximum probability occurs when $\mathbf{v} = \mathbf{0}$. Hence, the min-entropy of $\mathbf{x}_i$ is

$$H_\infty(\mathbf{x}_i) = -\log_2(\Pr[\mathbf{x}_i = \mathbf{0}]) \geq \omega(\log m).$$

This simplifies to $\Pr[\mathbf{x}_i = \mathbf{0}] \leq 2^{-\omega(\log m)}$. i.e., Probability of getting the most likely value, $\mathbf{x}_i = \mathbf{0}$ is very low.

If $\mathbf{x}_i$ are sampled independently from the discrete Gaussian distribution $\mathcal{D}_{\mathcal{L},\zeta}$, then their sum $\mathbf{x} = \sum_{i=1}^{k} \mathbf{x}_i$ behaves like a Gaussian with a larger width parameter, ζ multiplied by $\sqrt{k}$. As k increases, the distribution of $\mathbf{x}$ becomes more spread out over the lattice, and therefore, the probability mass at any single point decreases. In particular, the probability that the sum equals $\mathbf{0}$ becomes negligible: $\Pr[\mathbf{x} = \mathbf{0}] = \mathsf{negl}(m)$. Hence, the sum of the vectors is non-zero with overwhelming probability.

Norm of $\mathbf{x}$: We now estimate the norm of $\mathbf{x}$. As discussed in Remark 7.1, we model each sample $\mathbf{x}_i$ as being drawn independently from a discrete Gaussian distribution with parameter ζ. Recall from Lemma 2.1, the expected squared Euclidean norm of a vector of dimension m sampled from a discrete Gaussian distribution is given by

$$\mathbb{E}\left[\|\mathbf{x}_i\|^2\right] \leq \left(\frac{1}{2\pi} + \frac{\epsilon}{1-\epsilon}\right)\zeta^2 m.$$

Since we are adding k such vectors write $\mathbf{x} = \sum_{i=1}^{k} \mathbf{x}_i$ and we have

$$\|\mathbf{x}\|^2 = \left\|\sum_{i=1}^{k} \mathbf{x}_i\right\|^2.$$

Expanding the squared norm and taking the expectation gives

$$\mathbb{E}\left[\|\mathbf{x}\|^2\right] = \sum_{i=1}^{k} \mathbb{E}\left[\|\mathbf{x}_i\|^2\right] + \sum_{i\neq j} \mathbb{E}\left[\langle \mathbf{x}_i, \mathbf{x}_j\rangle\right].$$

Because the samples are independent and the discrete Gaussian distribution is symmetric about the origin, the cross terms satisfy $\mathbb{E}[\langle \mathbf{x}_i, \mathbf{x}_j\rangle] = 0$, and we obtain

$$\mathbb{E}\left[\|\mathbf{x}\|^2\right] = \sum_{i=1}^{k} \mathbb{E}\left[\|\mathbf{x}_i\|^2\right] \leq k\left(\frac{1}{2\pi} + \frac{\epsilon}{1-\epsilon}\right)\zeta^2 m. \tag{7.1}$$

Recall that, Jensen's inequality for concave functions states that the function of an expected value is greater than or equal to the expected value of the function. Hence, applying Jensen's inequality (since the square-root function is concave) to Equation (7.1), we obtain

$$\mathbb{E}\left[\|\mathbf{x}\|\right] \leq \sqrt{\mathbb{E}\left[\|\mathbf{x}\|^2\right]} \leq \zeta\sqrt{km\left(\frac{1}{2\pi} + \frac{\epsilon}{1-\epsilon}\right)}.$$

For $\mathbf{x}_j + \mathbf{x}$ to be a valid solution to the rOM-ISIS problem, we require $\|\mathbf{x}_j + \mathbf{x}\| \leq \beta$, and so it suffices to choose parameters such that $\zeta\sqrt{(k+1)m\left(\frac{1}{2\pi} + \frac{\epsilon}{1-\epsilon}\right)} \leq \beta$.

Remark 7.2. *Since ζ is chosen above the smoothing parameter (i.e., $\zeta > \eta_\epsilon(\mathcal{L})$ with negligible ϵ), the term $\frac{\epsilon}{1-\epsilon}$ becomes negligible and may be omitted asymptotically. Therefore, we use the simplified estimate*

$$\mathbb{E}\left[\|\mathbf{x}\|\right] \leq \zeta\sqrt{\frac{(k+1)m}{2\pi}}.$$

For simplicity we assume that the final list contains a single solution to the k-sum problem. If there are many solutions, then the later part of the attack works even more effectively.

The Algorithm

We describe our algorithm for the rOM-ISIS problem in detail. Note that we follow the same notations that we used in the k-tree algorithm discussed in Chapter 6 for consistency. In particular, $k = 2^q$ is the number of lists, m_0 is the initial list length and m is the dimension of the vector.

To begin, we select the largest integer k, restricted to powers of two, such that the condition $\zeta\sqrt{\frac{(k+1)m}{2\pi}} \leq \beta$ is satisfied. This choice of k ensures the required norm bound is met for the resulting vector $\mathbf{x}$. We then compute the total number of required samples for the k-tree algorithm as $\tau = km_0$, and proceed by making τ preimage queries on the zero syndrome vector ($\mathbf{t} = \mathbf{0}$). These queries yield a list of $(\mathbf{x}_i, \mathbf{y}_i)$ pairs, and these $\mathbf{y}_i$'s serve as inputs to the k-list algorithm. The algorithm searches for a set of $\mathbf{y}_i$ vectors that sum to zero. Suppose the algorithm found the index set $\mathcal{I}$ of size k with $\sum_{i\in\mathcal{I}} \mathbf{y}_i = \mathbf{0}$. Upon finding such a set, we can find the corresponding $\mathbf{x}_i$'s and set $\mathbf{x} = \sum_{i\in\mathcal{I}} \mathbf{x}_i$ such that $\mathbf{Ax} = \mathbf{0}$. Note that $\mathbf{x}$ is non-zero with high probability and its expected norm is bounded by $\|\mathbf{x}\| \leq \zeta\sqrt{km/(2\pi)}$ as discussed above.

Next, we make an additional $\tau + 1$ syndrome queries to obtain challenge vectors $\mathbf{t}_j$ for $j = 1, \ldots, \tau + 1$, followed by corresponding preimage queries which yield pairs $(\mathbf{x}_j, \mathbf{y}_j)$

that satisfy the rOM-ISIS conditions $\mathbf{A}\mathbf{x}_j + \mathbf{B}\mathbf{y}_j = \mathbf{t}_j$. For each such triple $(\mathbf{x}_j, \mathbf{y}_j, \mathbf{t}_j)$, we generate an additional solution by computing $\mathbf{x}_j \pm \mathbf{x}$, and select the sign that minimizes the Euclidean norm of the vector. In total, this process outputs $2(\tau + 1)$ valid solution triples of the form $(\mathbf{x}_j, \mathbf{y}_j, \mathbf{t}_j)$ and $(\mathbf{x}_j \pm \mathbf{x}, \mathbf{y}_j, \mathbf{t}_j)$.

Algorithm 6 A Combinatorial Algorithm for rOM-ISIS Problem

Input: $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_p^{n \times m}$, $k = 2^q$ such that $\zeta\sqrt{\frac{(k+1)m}{2\pi}} \leq \beta$, m_0.
Output: $\{(\mathbf{x}_j, \mathbf{y}_j, \mathbf{t}_j)\}$ and $\{(\mathbf{x}_j \pm \mathbf{x}, \mathbf{y}_j, \mathbf{t}_j)\}$ for $1 \leq j \leq \tau + 1$

Set $\tau = km_0$
for $l = 1, \ldots, k$ **do**
    Initialise $L_l \leftarrow \emptyset$.
    for $i = 1, \ldots, m_0$ **do**
        $(\mathbf{x}_{l,i}, \mathbf{y}_{l,i}) \leftarrow \mathsf{PreimageQuery}(\mathbf{t} = \mathbf{0})$
        Append $\mathbf{y}_{l,i}$ to L_l
    end for
end for
Run the *k-list algorithm* on $L_1, \ldots, L_k$. ▷ see Algorithm 5
Let $(\mathbf{y}_{1,i_1}, \mathbf{y}_{2,i_2}, \ldots, \mathbf{y}_{k,i_k})$ be such that $\sum_{l=1}^{k} \mathbf{y}_{l,i_l} = \mathbf{0}$
Set $\mathbf{x} \leftarrow \sum_{l=1}^{k} \mathbf{x}_{l,i_l}$
for $j = 1, \ldots, \tau + 1$ **do**
    $\mathbf{t}_j \leftarrow \mathsf{SyndromeQuery}$
    $(\mathbf{x}_j, \mathbf{y}_j) \leftarrow \mathsf{PreimageQuery}(\mathbf{t}_j)$
end for
return $\{(\mathbf{x}_j, \mathbf{y}_j, \mathbf{t}_j)\}$ and $\{(\mathbf{x}_j \pm \mathbf{x}, \mathbf{y}_j, \mathbf{t}_j)\}$ for $1 \leq j \leq \tau + 1$

The total number of preimage queries made is $\tau + \tau + 1$ and the number of outputs is $2(\tau + 1)$. So the attack wins as long as all vectors satisfy the required bounds.

The choice of suitable k is to trade-off the success probability and the running time. The vector $\mathbf{x}$ is a sum of k vectors, $\mathbf{x}_i$ and so the expected value of $\|\mathbf{x}\|$ is $\zeta\sqrt{km/(2\pi)}$. For the algorithm to succeed we need at least one of $\|\mathbf{x}_j \pm \mathbf{x}\| \leq \beta$ for every j. The probability this happens depends on how much of a gap there is between $\zeta\sqrt{km/(2\pi)}$ and β.

So the tension is between ensuring the success probability is acceptable and also choosing k, according to Figure 6.3, to lower the running time. In practice, we will choose the parameters so that there are several solutions $\mathbf{x}$ coming from the final list, to increase the chances that, for a given $\mathbf{x}_j$, one of our $\mathbf{x}$ vectors is such that $\|\mathbf{x}_j \pm \mathbf{x}\| \leq \beta$.

Remark 7.3. *Instead of placing the preimage query results into k independent lists, as we have done, one could alternatively create a single sufficiently large list L and search for vectors $\mathbf{y}_i \in L$ whose sum is zero. This can be achieved by duplicating the list and performing self-merging at each level, as discussed in [LF06, TSG25]. Unlike in the*

binary-vector XOR setting, for vectors in $\{-1, 1\}$ *we do not need to worry about trivial merges, and we expect the complexity analysis to remain essentially the same as in the* k*-list version. Moreover, depending on the list size, one might be able to reduce the number of required preimage queries, but by duplicating, one looses the independency between lists. A detailed complexity analysis of such a single-list approach would be an interesting direction for future work.*

7.2.1 Impact on the NIBS scheme

The algorithm described in Section 7.2 successfully constructs a solution to the rOM-ISIS problem. However, a potential concern may arise regarding the reuse of certain target syndromes during the generation of multiple valid preimage tuples.

In our approach, among the $\ell + 1$ required preimage solutions, the first tuple $(\mathbf{x}_j, \mathbf{y}_j, \mathbf{t}_j)$ is obtained directly from the preimage oracle. Subsequently, an additional valid solution $(\mathbf{x}, \mathbf{y}, \mathbf{t})$ is constructed using a combination of vectors obtained via the k-sum algorithm technique. While the $\mathbf{x}$ components of these tuples are distinct, both tuples share the same $\mathbf{y} = \mathbf{y}_j$ and target vector $\mathbf{t} = \mathbf{t}_j$.

This naturally leads to the following question: What is the effect of having two valid preimage tuples that share the same $\mathbf{y}$ and $\mathbf{t}$ on the overall security of the blind signature scheme? More specifically, does the repetition of the $\mathbf{y}$ and $\mathbf{t}$ components lead to a real attack on the scheme?

To address this question, we must examine the construction of the blind signature scheme and understand how the rOM-ISIS assumption is embedded within the security proof. For consistency in notation and readability of the thesis, we adopt a different set of symbols than those used in the original paper by Baldimtsi et al. [BCGY24].

NIBS scheme:

In the design of the new non-interactive blind signature (NIBS) scheme, the signer and receiver operate over public matrices $\mathbf{A}, \mathbf{B}, \mathbf{M} \in \mathbb{Z}_p^{n \times 2m}$. To generate a presignature, the signer first samples a random vector $\mathbf{y} \in \{\pm 1\}^{2m}$ and computes a preimage for the modified syndrome vector $(\mathsf{pk}_R - \mathbf{By})$, where pk_R denotes the receiver's public key. An important observation at this point is the increase in vector dimension to $2m$.

The receiver's public key pk_R is defined as

$$\mathbf{t} = \mathbf{Mz} + \mathsf{H}(\delta),$$

where $\delta \leftarrow\$ \{0,1\}^\lambda$ and $\mathbf{z} \leftarrow\$ \mathcal{D}_{\mathbb{Z}_q^{2m},\zeta/m}$ are receiver's secret keys. So the signer samples a short vector $\mathbf{x}$ such that:

$$\mathbf{A}\mathbf{x} = \mathsf{pk}_R - \mathbf{B}\mathbf{y}.$$

Given the presignature and nonce pair $(\mathbf{x}, \mathbf{y})$ the receiver creates a non-interactive zero-knowledge (NIZK) proof π attesting to knowledge of valid witnesses. The proof asserts that, given the inputs $\mathbf{A}, \mathbf{B}, \mathbf{M}, \mathbf{w}, \delta$, there exist vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ satisfying the following conditions:

$$\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} = \mathbf{M}\mathbf{z} + \mathsf{H}(\delta) \quad \wedge \quad \mathbf{w} = \mathbf{M}\begin{bmatrix}\mathbf{z}_\perp \\ \mathbf{x}_\perp\end{bmatrix},$$

$$\mathbf{y} = [y_1, y_2, \ldots, y_{2m}]^T, \quad \text{with } y_i \in \{\pm 1\} \text{ for all } i, \quad \|\mathbf{z}\| \leq \frac{\sqrt{2}\beta}{m}, \quad \|\mathbf{x}\| \leq \sqrt{2}\beta.$$

The $\sqrt{2}$ factor in the norm bound arises from the increase in vector dimension to $2m$. Finally, the receiver sets $\mu := (\mathbf{w}, \delta)$ as the message and $\sigma := (\pi, \mathsf{c})$ as the corresponding signature where c is an encryption of $\mathbf{x}\|\mathbf{y}\|\mathbf{z}$ using some randomness $r \leftarrow\$ \{0,1\}^\lambda$. To verify a signature, one simply runs the verification algorithm for the NIZK.

Effect on the One-more unforgeability:

Baldimtsi et al. use the rOM-ISIS assumption to prove the one-more unforgeability of their protocol. To win the one-more unforgeability game, the adversary must output $\ell+1$ message and signature pairs $(\mu_i, \sigma_i)_{i\in[\ell+1]}$ such that $\mu_i \neq \mu_j$ for $1 \leq i < j \leq \ell + 1$ and on input the verification key $\mathsf{vk} = \mathbf{A}$, μ_i, and σ_i, the verification algorithm accepts and outputs 1 for all $i \in [\ell + 1]$. The message is $\mu = (\mathbf{w}, \delta)$ where $\mathbf{w}$ and δ are defined as above, and the signature is $\sigma = (\pi, \mathsf{c})$. For the full construction of the NIBS scheme refer to Section 7 of [BCGY24].

Suppose an adversary who follows the above attack strategy outputs two solutions with the same $\mathbf{t}$ and $\mathbf{y}$ for the rOM-ISIS assumption (say $\mathbf{t}_i = \mathbf{t}_j$, and $\mathbf{y}_i = \mathbf{y}_j$). Then according to the above definition of $\mathbf{t}$, this means $\delta_i = \delta_j$ and $\mathbf{z}_i = \mathbf{z}_j$. However, since we have different presignatures ($\mathbf{x}_i \neq \mathbf{x}_j$), this implies that $\mathbf{w}_i \neq \mathbf{w}_j$ and therefore, $\mu_i \neq \mu_j$. Moreover, having same $\mathbf{t}$ or $\mathbf{y}$ does not affect the validity of an instance $(\mathbf{A}, \mathbf{B}, \mathbf{M}, \mathsf{pke.pk}, \mathsf{c}_i, \mathbf{w}_i, \delta_i)$ and therefore the verification algorithm accepts and output 1 for all $i \in [\ell + 1]$. Therefore, we conclude that it is allowed to have the same $\mathbf{t}$ and $\mathbf{y}$ in the $\ell + 1$ distinct solutions and the adversary can win the game.

The NIBS construction provides both the value of β and the norm bound that the short vector $\mathbf{z}$ should satisfy in this scheme. According to this construction $\beta = \zeta\sqrt{m}$ and $\|\mathbf{x}\| \leq \sqrt{2}\beta$. For our resulting vector $\mathbf{x}$, which has dimension $2m$, the expected norm is given by

$\zeta\sqrt{\frac{2m(k+1)}{2\pi}}$. This expected norm must satisfy the given norm bound. Therefore, we can determine the number of lists k that can be used while satisfying this condition. To ensure that the norm bound is met, the number of lists should be small with $k \leq 2\pi - 1 \approx 5$. Since we consider k to be powers of 2, this sets $k = 4$. With $m = 1024$, solving a $4-$list algorithm will be infeasible with a cost $\mathcal{O}(2^{426})$ as shown in Figure 6.3 and the running time increases further when the vectors are of length $2m$. Hence, we conclude that the NIBS scheme remains secure for the particular parameter set chosen in [BCGY24].

7.3 A way to choose different t's

We have given an algorithm that solves the rOM-ISIS problem, and have explained its effect on cryptanalysis of the NIBS scheme given in [BCGY24]. But it is also a natural question to find an algorithm that solves rOM-ISIS under the additional requirement that all the vectors $\mathbf{t}$ in the output are distinct. We stress that this is not required in the rOM-ISIS problem as stated, nor in the scheme. But it is a natural question to ask and we were able to make some progress on it.

The method we are proposing is similar to the combinatorial attack strategy given in [BCGY24] with a few tweaks that allow us to use a tree-like algorithm that we discuss in the next chapter. The attack is performed in two phases. We give the steps for the first phase of the attack as follows.

Phase I:

The attacker first makes $\tau+1$ syndrome queries, where τ is a parameter to be specified later. Let S denote the set of received challenge vectors:

$$S = \{\mathbf{t}' \ : \ \mathbf{t}' \leftarrow\!\$\ \mathbb{Z}_p^n\}, \qquad |S| = \tau + 1.$$

Now, construct the set $T = \{a_i\mathbf{e}_i \mid \forall i \in [n], a_i \in \mathbb{Z}_p\}$, where $\mathbf{e}_i$ are the canonical basis vectors of $\mathbb{Z}_p^n$. This set consists of all scalar multiples of the standard basis vectors, resulting in a total size of np. The attacker then proceeds to query the preimage oracle for every element $a_i\mathbf{e}_i \in T$. Since T contains np elements, the total number of queries made to the oracle is exactly np. For each query, the oracle returns a pair $(\mathbf{x}_{i,a_i}, \mathbf{y}_{i,a_i})$ such that the equation $\mathbf{A}\mathbf{x}_{i,a_i} + \mathbf{B}\mathbf{y}_{i,a_i} = a_i\mathbf{e}_i$ holds. Moreover, the vector $\mathbf{x}_{i,a_i}$ is bounded in norm by $\|\mathbf{x}_{i,a_i}\| \leq \zeta\sqrt{m}$, and the vector $\mathbf{y}_{i,a}$ lies in the set $\{\pm 1\}^m$.

Note that any vector $\mathbf{t}' \in \mathbb{Z}_p^n$ can be uniquely expressed as a linear combination of basis vectors, namely, $\mathbf{t}' = \sum_{i=1}^n a_i\mathbf{e}_i$ for some coefficients $a_i \in \mathbb{Z}_p$. In other words, any target

vector $\mathbf{t}' \in S$ can be written with at most n entries from T. Therefore, the attacker can obtain the following.

$$\mathbf{A}\sum_{i=1}^{n}\mathbf{x}_{i,a_i} + \mathbf{B}\sum_{i=1}^{n}\mathbf{y}_{i,a_i} = \sum_{i=1}^{n} a_i\mathbf{e}_i = \mathbf{t}'. \tag{7.2}$$

Defining $X = \sum_{i=1}^{n}\mathbf{x}_{i,a_i}$ and $Y = \sum_{i=1}^{n}\mathbf{y}_{i,a_i}$, the attacker gets $\mathbf{A}X + \mathbf{B}Y = \mathbf{t}'$ where X, Y are integer vectors. We stress that $Y \notin \{-1, 1\}^m$ and therefore is not a valid solution to the rOM-ISIS assumption. In this phase we found a way to write any $\mathbf{t}' \in S$ in the form $\mathbf{A}X + \mathbf{B}Y$. However, Y is still not in the right form and this leads to the next phase.

Algorithm 7 Phase I: Reformulate $\mathbf{t}' \in S$

Input: $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_p^{n\times m}, S = \{\mathbf{t}' \mid \mathbf{t}' \leftarrow\$ \mathsf{SyndromeQuery}()\}$
Output: (X, Y) such that $\mathbf{A}X + \mathbf{B}Y = \mathbf{t}'$

for $i = 1, \ldots, n$ **do**
  for $a = 1, \ldots, p-1$ **do**
    $(\mathbf{x}_{i,a}, \mathbf{y}_{i,a}) \leftarrow \mathsf{PreimageQuery}(a\mathbf{e}_i)$
  end for
end for
for $\mathbf{t}' \leftarrow S$ **do**
  Compute $\mathbf{t}' = \sum_{i=1}^{n} a\mathbf{e}_i$
  Get $\mathbf{A}\sum_{i=1}^{n}\mathbf{x}_{i,a} + \mathbf{B}\sum_{i=1}^{n}\mathbf{y}_{i,a} = \mathbf{t}'$
  Set $X = \sum_{i=1}^{n}\mathbf{x}_{i,a}$ and $Y = \sum_{i=1}^{n}\mathbf{y}_{i,a}$.
  return X, Y
end for

Phase II:

In the second phase of the attack, we make additional queries to the preimage oracle, specifically targeting the zero syndrome vector $\mathbf{t} = \mathbf{0}$. This phase mirrors the preimage query strategy used earlier, but here we have a different goal.

To begin, the attacker issues multiple preimage queries with $\mathbf{t} = \mathbf{0}$ to the oracle. Say the attacker made N such preimage queries. Each oracle response consists of a pair $(\mathbf{x}_j, \mathbf{y}_j)$ satisfying the relation

$$\mathbf{A}\mathbf{x}_j + \mathbf{B}\mathbf{y}_j = \mathbf{0},$$

with constraints $\|\mathbf{x}_j\| \leq \zeta\sqrt{m}$ and $\mathbf{y}_j \in \{\pm 1\}^m$ for each j. These vectors form a pool of preimages whose combination can help compute a final valid solution to the rOM-ISIS problem for each (X, Y) from Phase I.

The attacker's objective is now to identify a subset $\mathcal{J}$ of indices such that the corresponding $\mathbf{y}_j$ vectors sum up to a target vector Y, where Y is the vector defined in Phase I as the sum of the selected vectors $\mathbf{y}_{i,a_i}$. Formally, find $\mathbf{y}_j$ such that

$$Y = \sum_{j \in \mathcal{J}} \mathbf{y}_j.$$

Substituting this relation back into the preimage equation yields

$$\mathbf{A} \sum_{j \in \mathcal{J}} \mathbf{x}_j + \mathbf{B}Y = \mathbf{0},$$

which simplifies to

$$\mathbf{B}Y = -\mathbf{A} \sum_{j \in \mathcal{J}} \mathbf{x}_j.$$

At this point, we combine this new information with the partial result obtained from Phase I. Recall from Phase I that we had a pair (X, Y) such that

$$\mathbf{t}' = \mathbf{A}X + \mathbf{B}Y.$$

Substituting the simplification from above into this expression gives

$$\mathbf{t}' = \mathbf{A}X + \mathbf{B}Y = \mathbf{A}X - \mathbf{A} \sum_{j \in \mathcal{J}} \mathbf{x}_j = \mathbf{A} \left(X - \sum_{j \in \mathcal{J}} \mathbf{x}_j \right).$$

This is summarised in the following equation

$$\mathbf{t}' = \mathbf{A} \left(X - \sum_{j \in \mathcal{J}} \mathbf{x}_j \right). \tag{7.3}$$

To produce a complete solution to the rOM-ISIS problem, we choose an unused preimage response for $\mathbf{t} = \mathbf{0}$ out of N queries made. Suppose this response was the pair $(\mathbf{x}_1, \mathbf{y}_1)$. We now combine this additional pair with the equation above, obtaining

$$\mathbf{t}' = \mathbf{A} \left(\mathbf{x}_1 + X - \sum_{j \in \mathcal{J}} \mathbf{x}_j \right) + \mathbf{B}\mathbf{y}_1.$$

Define

$$\mathbf{x}' := \mathbf{x}_1 + X - \sum_{j \in \mathcal{J}} \mathbf{x}_j \quad \text{and} \quad \mathbf{y}' := \mathbf{y}_1.$$

Then, we have

$$\mathbf{t}' = \mathbf{A}\mathbf{x}' + \mathbf{B}\mathbf{y}',$$

which satisfies the structure of a rOM-ISIS solution.

Up to this stage, we have made $\tau = np+N$ preimage queries to obtain a triple $(\mathbf{x}', \mathbf{y}', \mathbf{t}')$ with the following properties:

- $\mathbf{x}'$ is an integer vector (a sum of $n+1+|\mathcal{J}|$ short vectors, each of norm at most $\zeta\sqrt{m}$),
- $\mathbf{y}' \in \{\pm 1\}^m$,
- $\mathbf{t}' \in S$,
- and the equation $\mathbf{t}' = \mathbf{A}\mathbf{x}' + \mathbf{B}\mathbf{y}'$ holds.

Any challenge vector $\mathbf{t}' \in S$ can be written in the required form using the pool of τ preimage vectors obtained during Phase I and Phase II. In particular, for $|S| = \tau + 1$, every $\mathbf{t}' \in S$ can be expressed as $\mathbf{t}' = \mathbf{A}\mathbf{x}' + \mathbf{B}\mathbf{y}'$, with $\mathbf{x}' \in \mathbb{Z}_p^m$ and $\mathbf{y}' \in \{-1, 1\}^m$ and therefore, the attacker can generate $\tau + 1$ valid rOM-ISIS solutions.

To analyse the quality of these $\tau + 1$ solutions, especially in terms of the norm of $\mathbf{x}'$, recall that each $\mathbf{x}_j$ is sampled from a discrete Gaussian distribution with standard deviation ζ. The expected Euclidean norm of such a vector is approximately $\zeta\sqrt{m/(2\pi)}$. Since $\mathbf{x}'$ is formed by adding together $(n+1+|\mathcal{J}|)$ such independent vectors, the expected norm scales linearly with the number of summands. Thus, the expected norm of $\mathbf{x}'$ is approximately:

$$\mathbb{E}[\|\mathbf{x}'\|] = \zeta\sqrt{(n+1+|\mathcal{J}|)\frac{m}{2\pi}}.$$

In order for $(\mathbf{x}', \mathbf{y}', \mathbf{t}')$ to be accepted as a valid solution to the rOM-ISIS instance, it is required that $\|\mathbf{x}'\| \leq \beta$, where β is the bound set by the problem definition. This concludes the attack on the rOM-ISIS problem with distinct challenge vectors $\mathbf{t}$'s.

Relation to a New Variant of the k-sum Problem: The core computational step in Phase II is to find a subset $\mathcal{J}$ such that $\sum_{j\in\mathcal{J}} \mathbf{y}_j = Y$. This defines a new variant of the classic k-*sum* problem: given a target vector $Y \in \mathbb{Z}_p^m$, which is itself a sum of n $\{\pm 1\}^m$ vectors, determine whether it can be reconstructed as the sum of other $\{\pm 1\}^m$ vectors.

This will be the focus of the next chapter (see Chapter 8), where we formally introduce and analyse this novel variant. In particular:

- The number of vectors that must be summed to form Y (i.e., the size of $\mathcal{J}$) determines the number of lists used in our algorithm.

- If Y is the sum of n vectors, then solving the problem will typically require $k = n$ lists to successfully reconstruct it.

As a consequence, the total number of short vectors contributing to $\mathbf{x}'$ become $(n + |\mathcal{J}| + 1) \approx 2n + 1$ on average. Therefore, the expected norm of $\mathbf{x}'$ is roughly:

$$\mathbb{E}[\|\mathbf{x}'\|] = \zeta\sqrt{(2n+1)\frac{m}{2\pi}},$$

which must still satisfy $\|\mathbf{x}'\| \leq \beta$ to yield a valid solution to the rOM-ISIS problem.

In the rOM-ISIS attack strategy, the target vector Y is expressed as a sum of at most n vectors, and the work of [BCGY24] sets $n = 512$. Therefore, recovering Y can be modelled as a generalised k-sum problem with $k = 2^9$ lists. Under this setting, the cost of finding a matching combination is approximately $\mathcal{O}(2^{0.31m})$ up to logarithmic factors (See Chapter 8 for further details).

It is important to note that this cost only accounts for the second phase of the attack. The overall complexity also depends on the cost of Phase I, which involves making preimage queries for vectors in the set T, requiring approximately $\mathcal{O}(np)$ queries. Thus, the total computational complexity of the complete attack strategy is given by $\mathcal{O}(\max(km_0, np))$ where m_0 is the initial list length for the k-sum problem.

7.4 Conclusion & Future Work

In this chapter, we examined two variants of the Inhomogeneous Short Integer Solution (ISIS) problem that have recently attracted attention in cryptographic research. Our primary focus was the randomised one-more (rOM-ISIS) problem, which underpins a non-interactive blind signature scheme proposed by Baldimtsi et al.

We proposed a novel combinatorial attack strategy targeting the rOM-ISIS problem. This strategy is based on a newly introduced Wagner-like k-tree algorithm for $\{-1, 1\}^m$-vectors, which extends traditional meet-in-the-middle techniques to more structured and efficient search trees. The goal was to investigate whether this approach could produce additional valid solutions to rOM-ISIS instances by exploiting the additive structure of the problem.

Our findings indicate that the answer is affirmative: using k-tree algorithms, one can successfully generate additional valid solutions, thereby posing a potential threat to both the rOM-ISIS assumption and the associated blind signature scheme. However, our cryptanalytic strategy needs more than polynomially-many queries to the oracles and has exponential complexity in the dimension m of the underlying vector space. Consequently,

no probabilistic polynomial-time (PPT) adversary can carry out this attack. Moreover, due to the carefully chosen parameters proposed by Baldimtsi et al. [BCGY24], the practical impact of the attack remains limited, and the scheme is secure under its current parameter instantiation.

Baldimtsi et al. explain in Section 2.5 of [BCGY24] that the goal of introducing the rOM-ISIS assumption is to strengthen the original OM-ISIS assumption against linear-combination attacks, aiming for a more robust assumption that resists any efficient or practical attacks regardless of parameter choices. In this chapter, we presented a combinatorial algorithm that improves the state-of-the-art for the security of the rOM-ISIS assumption proposed by Baldimtsi et al. Thus, we argue that the new structure of the rOM-ISIS allows new types of attacks like the k-sum algorithm we proposed that are not applicable on the original problem and therefore the rOM-ISIS assumption may falls short of its intended security goals.

In conclusion, while our proposed attack does not currently undermine the security of the PPT scheme under the given parameters, it provides a new perspective on the rOM-ISIS problem's hardness. We believe that our combinatorial framework lays the groundwork for future cryptanalysis and can help guide parameter selection and security evaluations in related cryptographic constructions.

Chapter 8

A k-sum Algorithm For a Given Vector

Contents

In Chapter 6, we introduced a new variant of the generalised birthday problem where the elements involved are vectors in $\{-1, 1\}^m$. In that setting, the objective was to find a collection of such vectors, chosen uniformly and independently at random, whose sum is the zero vector. We analysed this problem and proposed a tree-based algorithm that achieves better complexity than the classical meet-in-the-middle approach. This algorithm was then applied to solve a new cryptographic assumption in Chapter 7. During this application, we observed that the core problem admits a natural generalisation, which forms the basis for the variant we study in this chapter.

The problem considered here can be viewed as a generalisation of Problem 6.2 from Chapter 6. As before, the inputs are uniformly random vectors from the set $\{-1, 1\}^m$. However, unlike the previous case where the target was the zero vector, we now consider a target vector $Y \in \mathbb{Z}^m$, which is a sum of k independently sampled ± 1 vectors. A formal definition for the new k-sum problem is given below. The addition is the usual component-wise vector addition of $\{-1, 1\}^m$ vectors.

Definition 8.1. *Given k vectors $\mathbf{y}_i \in \{-1, 1\}^m$, let $Y = \sum_{i=1}^{k} \mathbf{y}_i$. Given k lists $L_1, \ldots, L_k$ of vectors drawn uniformly and independently at random from $\{-1, 1\}^m$, find $\mathbf{x}_1 \in L_1, \ldots, \mathbf{x}_k \in L_k$ such that $\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_k = Y$.*

This seemingly small modification significantly alters the structure and complexity of the problem, and calls for an adapted algorithmic approach.

8.1 Understanding the Core Problem

Before designing an efficient algorithm, it is important to understand the inherent hardness of the problem and the type of complexity one might expect. In this variant, the goal is to find a sum of random ± 1 vectors that equals a fixed vector $Y \in \mathbb{Z}^m$. The fixed nature of Y significantly complicates the problem, since the probability that a sum of random vectors (which approximately follows a normal distribution) matches a specific target depends on this target Y. As a result, standard collision-based complexity estimates such as those in [Sel95] are not directly applicable.

In general, the collision-based techniques assume that both sides of the equality are independent random variables drawn from the same distribution. However, in our setting, the target vector $Y = \sum_{i=1}^{k} \mathbf{y}_i$ is fixed and determined by known input vectors $\mathbf{y}_i \in \{-1, 1\}^m$. The task is to find vectors $\mathbf{x}_i \in L_i$, where each L_i contains independently and uniformly chosen random vectors, such that $\sum_{i=1}^{k} \mathbf{x}_i = Y$. Since Y is not a random variable, it is incorrect to model this as a symmetric collision event between two random sums. Therefore, the standard technique of computing the collision probability as the sum of squared probabilities does not apply for a given instance.

To further illustrate this, consider a basic meet-in-the-middle approach. Suppose the target vector Y is the sum of $k = 4$ random vectors $\mathbf{y}_i \in \{-1, 1\}^m$, and we are given four lists $L_1, \ldots, L_4$ of random vectors. The task is to find vectors $\mathbf{x}_i \in L_i$ such that $\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathbf{x}_4 = Y$.

We define two intermediate sum lists,

$$L_{12} = \{\mathbf{x}_1 + \mathbf{x}_2 : \mathbf{x}_1 \in L_1, \mathbf{x}_2 \in L_2\} \quad \text{and} \quad L_{34} = \{Y - (\mathbf{x}_3 + \mathbf{x}_4) : \mathbf{x}_3 \in L_3, \mathbf{x}_4 \in L_4\}.$$

A solution is found when an element from L_{12} matches an element from L_{34}. However, when Y is fixed, the distributions of the vectors in L_{12} and L_{34} are no longer identical. In particular, L_{12} consists of sums of independent random vectors, while L_{34} is offset by the fixed target Y, which introduces dependence. As a result, these two lists are not symmetric random variables, and the standard assumptions behind collision probability estimates no longer hold.

Instead, we employ a heuristic based on the birthday paradox to obtain a lower bound on the complexity of the problem. The key idea is that the sum of random ± 1 vectors $Y = \sum_{i=1}^{k} \mathbf{y}_i$ in each coordinate behaves like a one-dimensional random walk, which can be approximated by a normal distribution. After k steps, each coordinate has mean 0 and variance k, so the typical size of a coordinate is on the order of $\sqrt{k}$. As a simplifying heuristic, we therefore approximate $Y \approx (\sqrt{k}, \ldots, \sqrt{k})$. While this assumption oversimplifies the true distribution of Y, we believe it suffices for a rough complexity estimate.

Let P_Y denote the probability that a sum of $k = 4$ random ± 1 values in a single coordinate equals $\sqrt{k} = 2$. This follows a binomial distribution:

$$P_Y = 2^{-k} \binom{k}{\frac{k+\sqrt{k}}{2}} = 2^{-4} \binom{4}{3} = 0.25.$$

Since $Y \in \mathbb{Z}^m$, the probability of matching all m coordinates simultaneously is $P_Y^m = (0.25)^m$. Now, suppose we have two lists of size $|L| = |R| = N$. Then the probability that at least one collision occurs is:

$$\Pr(\text{a match occurs}) = 1 - (1 - P_Y^m)^{N^2}.$$

Applying the standard approximation $(1 - x)^a \approx e^{-ax}$ for small x, we get

$$\Pr(\text{a match occurs}) \approx 1 - e^{-N^2 \cdot P_Y^m}.$$

To ensure a success probability of at least $1/2$, we require:

$$e^{-N^2 \cdot P_Y^m} < \frac{1}{2} \quad \Longrightarrow \quad N > \sqrt{\ln 2} \cdot P_Y^{-m/2}.$$

Since $P_Y = 0.25 = 2^{-2}$, this implies $N > \sqrt{\ln 2} \cdot (2^m)$, and hence, the overall time and space complexity of the algorithm is $\mathcal{O}(2^m)$.

As the number of summands increases, the complexity of the meet-in-the-middle approach grows accordingly. For example, when $k = 16$ and $Y = (4, \ldots, 4)$, we have

$P_Y = 2^{-16}\binom{16}{10} = 0.1222$. This gives $N \approx 2.86^m$, and the algorithm runs in time $\mathcal{O}(2^{1.516m})$. If we instead consider the mean value, which is zero, then Y is the zero vector and the sum of four vectors yields $P_Y = 2^{-4}\binom{4}{2} = 0.375$. In this case $N \approx 2^{0.7075m}$. Thus, for each instance, the running time depends on the chosen target Y.

8.1.1 Motivation & Research Problem

The above analysis highlights that the fixed-target variant of the problem is significantly more difficult than both the binary case (where $\mathbf{x}_i \in \{0,1\}^m$) and the zero-sum case involving ± 1 vectors. The requirement to match a specific target vector substantially reduces the success probability of random sampling approaches, making the problem inherently more complex. Therefore, it is both necessary and intellectually compelling to investigate whether more advanced algorithmic techniques can be leveraged to reduce this complexity.

In the previous analysis of the zero-sum version with ± 1 vectors, we observed that tree-like algorithms led to a significant improvement over basic strategies such as meet-in-the-middle. This motivates us to consider whether similar algorithmic frameworks can offer gains in the fixed-target setting as well.

8.2 Generalised k-tree Algorithm

We treat this new problem as a generalisation of the zero-sum problem (Definition 6.2), and hence, the k-tree algorithm we design will broadly follow the same structure as the algorithm previously described for the k-tree problem in Chapter 6.

The primary difference lies in how we handle the partial sums at each level of the algorithm. We exploit the fact that the target vector Y is itself the sum of random ± 1 vectors, which provides structure to the expected distribution of the intermediate sums. In particular, instead of merging vectors from the lists with the goal of reaching the zero vector, we now merge them to match specific partial sums of the vectors $\mathbf{y}_i$.

This shift in target affects the calculation of the success probability, as the distributional properties of the intermediate sums differ from those in the zero-sum case. However, this modification does not alter the overarching strategy of the algorithm. As a result, many of the insights, and analytical techniques developed for the k-tree algorithm (Algorithm 5) remain valid and applicable in this more general setting.

One feature of the k-tree algorithm is that the lengths of the intermediate lists those generated at each merging stage are not necessarily fixed. However, for the purpose of our

asymptotic analysis, we model the behaviour of the algorithm using the *expected values* of the lengths of these intermediate lists. This modelling choice simplifies the analysis while preserving accuracy in the average-case performance estimates.

We adopt the same analytical framework as in Section 6.3.2, which assumes that the expected lengths of all intermediate lists are equal to the initial list length m_0. This assumption allows us to obtain tractable expressions for the running time of the algorithm. Moreover, in Section 8.3.1, we will provide empirical evidence to support the validity of this assumption in practice.

Notation:

The following notation holds implicitly throughout the chapter. We assume $k = 2^q$. $Y = \sum_{i=1}^{k} \mathbf{y}_i$ where $\mathbf{y}_i \in \{-1, 1\}^m$ are chosen uniformly and independently at random. Similar to the previous discussions, $\mathbf{x}_i^{(j)} \in L_i^{(j)}$ represent any vector $\mathbf{x}$ from the i^{th} list at the j^{th} level of the k-tree where $i = 1, \ldots, k/2^j$ and $j = 1, \ldots, q$. The symbol $\|$ denotes the concatenation of vectors, e.g., $\pi_1^{(1)} = \left(\pi_1^{(0)} \parallel \pi_2^{(0)}\right) = \left(\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}\right)$.

8.2.1 Overview of the Algorithm

We now illustrate the analysis using an instance of the k-list problem with $k = 8$, and refer the reader to Figure 8.1 for a visual overview of the algorithmic structure.

We are given eight initial lists $L_1^{(0)}, \ldots, L_8^{(0)}$, each of size m_0.

$$L_i^{(0)} = \left\{ \left(\mathbf{x}_i^{(0)}, \pi_i^{(0)}\right) \;\middle|\; \mathbf{x}_i^{(0)} \hookleftarrow\$ \{-1, 1\}^m \right\}.$$

We define, $\pi_i^{(0)} = \mathbf{x}_i^{(0)}$. The task is to find vectors $\mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_8^{(0)}$ such that their sum equals the target vector, $Y = \sum_{i=1}^{8} \mathbf{y}_i$, where each $\mathbf{y}_i$ is a random vector with entries in $\{\pm 1\}$.

Level 1. At the first level, we merge the lists pairwise. For example, to construct $L_1^{(1)}$, we consider all pairs of vectors from $L_1^{(0)} \times L_2^{(0)}$ such that their sum matches the first l_1 coordinates of the corresponding partial sum $\mathbf{y}_1 + \mathbf{y}_2$. We denote this matching condition as $(\mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)})_{l_1} = (\mathbf{y}_1 + \mathbf{y}_2)_{l_1}$.

$$L_1^{(1)} = \left\{ \left(\mathbf{x}_1^{(1)}, \pi_1^{(1)}\right) \;\middle|\; \begin{array}{l} (\mathbf{x}_1^{(0)}, -) \in L_1^{(0)}, \quad (\mathbf{x}_2^{(0)}, -) \in L_2^{(0)}, \\ \mathbf{x}_1^{(1)} = \mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)}, \left(\mathbf{x}_1^{(1)}\right)_{l_1} = (\mathbf{y}_1 + \mathbf{y}_2)_{l_1} \\ \pi_1^{(1)} = \left(\pi_1^{(0)} \parallel \pi_2^{(0)}\right) \end{array} \right\}.$$

In the definition above, we use $(\mathbf{x}_1^{(0)}, -) \in L_1^{(0)}$ and $(\mathbf{x}_2^{(0)}, -) \in L_2^{(0)}$ to indicate that we only take the *first entry* of each pair in the list, i.e., the vector itself, in order to compute the sum. The back-pointer, $\pi_i^{(j)}$ is preserved in the new pair that records how this sum was formed.

Similarly, we generate the lists $L_2^{(1)}, L_3^{(1)}, L_4^{(1)}$ by merging $L_3^{(0)}$ with $L_4^{(0)}$, $L_5^{(0)}$ with $L_6^{(0)}$, and $L_7^{(0)}$ with $L_8^{(0)}$, respectively.

Level 2. At the second level, we merge the lists from the previous level. For instance, to construct $L_1^{(2)}$, we consider pairs from $L_1^{(1)} \times L_2^{(1)}$ such that the sum of the corresponding vectors matches the next l_2 coordinates of $\mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3 + \mathbf{y}_4$. Recall that the first entry of the elements of $L_1^{(1)}$ are of the form $\mathbf{x}_1^{(1)} = \mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)}$, and that of $L_2^{(1)}$ are of the form $\mathbf{x}_2^{(1)} = \mathbf{x}_3^{(0)} + \mathbf{x}_4^{(0)}$, so their sum yields $\mathbf{x}_1^{(1)} + \mathbf{x}_2^{(1)} = \mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)} + \mathbf{x}_3^{(0)} + \mathbf{x}_4^{(0)}$. The corresponding matching condition is,

$$(\mathbf{x}_1^{(1)} + \mathbf{x}_2^{(1)})_{l_1+l_2} = (\mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3 + \mathbf{y}_4)_{l_1+l_2}.$$

Note that one only needs to check the last l_2 coordinates, since the first l_1 are already guaranteed to be correct from the previous step. In a similar manner, we generate $L_2^{(2)}$ from $L_3^{(1)} \times L_4^{(1)}$, matching against the l_2 coordinates of $\mathbf{y}_5 + \mathbf{y}_6 + \mathbf{y}_7 + \mathbf{y}_8$.

By construction, the list $L_1^{(2)}$ contains vectors whose first $l_1 + l_2$ coordinates match those of $\mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3 + \mathbf{y}_4$, and $L_2^{(2)}$ matches those of $\mathbf{y}_5 + \mathbf{y}_6 + \mathbf{y}_7 + \mathbf{y}_8$.

Level 3. Finally, in the third level, we add the first entries of $(\mathbf{x}_1^{(2)}, \pi_1^{(2)}) \in L_1^{(2)}$ and $(\mathbf{x}_2^{(2)}, \pi_2^{(2)}) \in L_2^{(2)}$ to generate the final list containing vectors $\mathbf{x}_1^{(3)} = \mathbf{x}_1^{(2)} + \mathbf{x}_2^{(2)}$, where

$$(\mathbf{x}_1^{(2)} + \mathbf{x}_2^{(2)})_{m-l_1-l_2} = \left(\sum_{i=1}^{8} \mathbf{y}_i \right)_{m-l_1-l_2}.$$

Since the vectors in $L_1^{(2)}$ and $L_2^{(2)}$ already satisfy the first $l_1 + l_2$ coordinates of the target Y, it suffices to check that their sum matches the remaining $m - l_1 - l_2$ coordinates. Each element in the final list, $L_1^{(3)}$ is of the form $(\mathbf{x}_1^{(3)}, \pi_1^{(3)})$ where $\pi_1^{(3)}$ has a chain of back pointers to the initial vectors $\mathbf{x}_i^{(0)} \in L_i^{(0)}$, ensuring that $\mathbf{x}_1^{(3)} = \sum_{i=1}^{8} \mathbf{x}_i^{(0)}$. Thus, the final list contains vectors that sum to the target vector Y, as desired.

Success Criterion. The goal is to ensure that the final list $L_1^{(q)}$ (with $q = \log_2 k = 3$ in this example) is non-empty. This translates to the condition

$$\mathbb{E}[|L_1^{(3)}|] \geq 1,$$

which provides a success guarantee in expectation. The parameters $l_1, l_2, \ldots, l_q$ and the initial list size m_0 must be chosen appropriately to satisfy this constraint. The algorithm will determine optimal values for these parameters so that the k-list problem admits a solution with high probability.

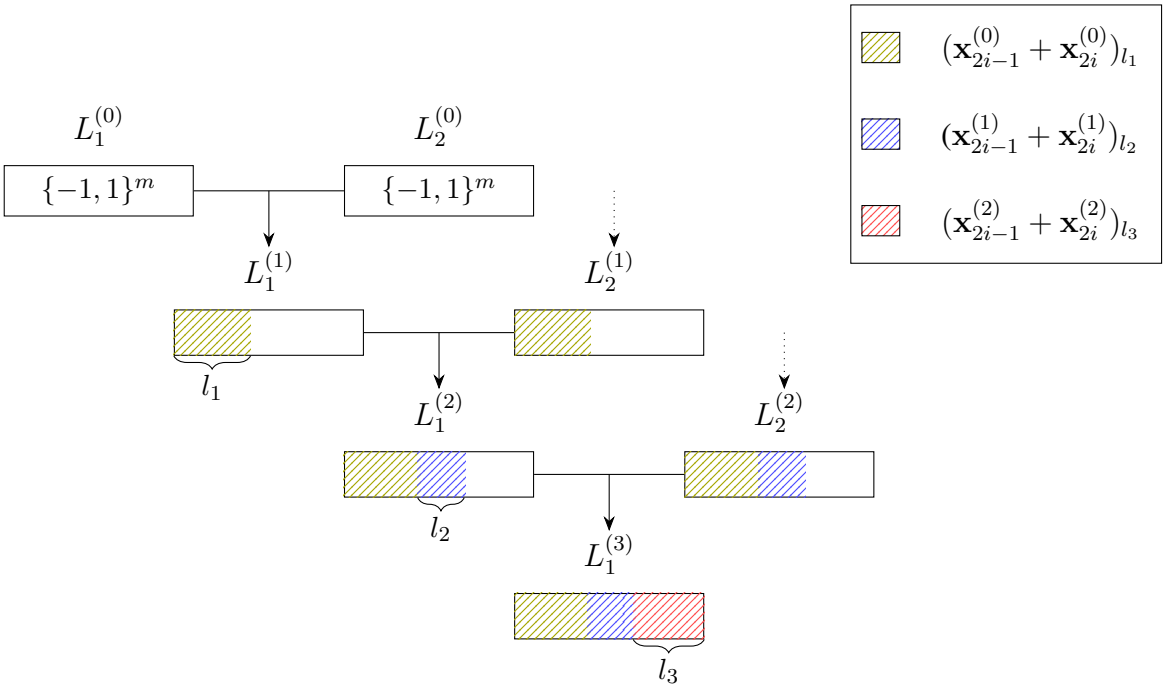


Figure 8.1: 8-list algorithm to find a set of vectors that sum to a given vector $Y = \sum_{i=1}^{8} \mathbf{y}_i$

8.2.2 Probability Calculation for a fixed Y

The main challenge in analysing this algorithm lies in computing the probability that a sum of vectors matches a corresponding sum of the $\mathbf{y}_i$. As discussed in Section 6.2.2, the sum of random $\{-1, 1\}$ vectors can be modelled as a random walk, and its distribution can be derived from properties of Pascal's triangle. When the number of steps increases, the distribution converges to a normal distribution. We again use this insight to estimate the success probabilities.

Recall that each vector $\mathbf{y}_i \in \{-1, 1\}^m$ is sampled independently and uniformly at random, and the target vector $Y = \sum_{i=1}^{k} \mathbf{y}_i$ is fixed and given to the algorithm. The distribution of Y and intermediate sums such as $\mathbf{y}_1 + \mathbf{y}_2$, or $\mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3 + \mathbf{y}_4$, follows the distribution given by Equation (6.3), which we restate below.

$$\Pr(X_n = p') = \begin{cases} 2^{-n}\binom{n}{\frac{n+p'}{2}} & \text{if } n + p' \text{ is even,} \\ 0 & \text{otherwise.} \end{cases} \tag{8.1}$$

Note that the parameter n in Equation (8.1) refers to the number of steps in the random walk at each level $j = 1, \ldots, q$ of the algorithm, and we take $n = 2^j$ as explained in Chapter 6.

Level 1. At level 1 of the algorithm, our goal is to match the first l_1 coordinates of the sum $\mathbf{y}_1 + \mathbf{y}_2$. Each coordinate in this sum takes a value in the set $\{-2, 0, 2\}$. Assuming $\mathbf{y}_1$ and $\mathbf{y}_2$ are uniformly random and independent vectors with entries in $\{-1, 1\}$, the distribution of their sum resembles a simple two-step random walk. For the analysis of a fixed Y, we take the following example Y with the expected distribution of values across the first l_1 coordinates as:

- $l_1/4$ entries equal to $+2$,
- $l_1/4$ entries equal to -2,
- $l_1/2$ entries equal to 0.

The lists $L_1^{(0)}$ and $L_2^{(0)}$ contain vectors $\mathbf{x}_1^{(0)}$ and $\mathbf{x}_2^{(0)}$, respectively, which are independent random vectors with entries in $\{-1, 1\}$. Let $\mathbf{x}_1^{(1)} = \mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)}$. We are interested in the probability that the first l_1 coordinates of $\mathbf{x}_1^{(1)}$ match those of $\mathbf{y}_1 + \mathbf{y}_2$.

Since $\mathbf{x}_1^{(1)}$ is a sum of independent random ± 1 vectors, we can assume that each coordinate of $\mathbf{x}_1^{(1)}$ is independently equal to:

- $+2$ with probability $\frac{1}{4}$,
- -2 with probability $\frac{1}{4}$,
- 0 with probability $\frac{1}{2}$.

Therefore, the probability that the first l_1 coordinates match the expected pattern in $\mathbf{y}_1 + \mathbf{y}_2$ is:

$$\Pr\left[(\mathbf{x}_1^{(1)})_{l_1} = (\mathbf{y}_1 + \mathbf{y}_2)_{l_1}\right] \approx \left(\frac{1}{4}\right)^{l_1/4} \cdot \left(\frac{1}{4}\right)^{l_1/4} \cdot \left(\frac{1}{2}\right)^{l_1/2} = \left(\frac{1}{2}\right)^{3l_1/2}.$$

Note that all the probabilities can be computed exactly for given $\{\mathbf{y}_i\}$. We will use this "average-case" behaviour for a fixed Y to estimate the success probability at each level of the algorithm. We emphasise that the following calculations are for this particular choice of Y.

General Setting. For the general case, we model the occurrence of each entry $p' = -n+2i$ in a vector using Equation (8.1). Accordingly, we expect approximately $\Pr(X_n = p') \cdot l_j$ occurrences of p' among the corresponding l_j coordinates of $\mathbf{y}$ on average.

Again, the probability that the sum of $\mathbf{x}_i^{(j)}$ has the same value as the corresponding sum of $\mathbf{y}_i$ is given by $\Pr(X_n = p')$ and it should satisfy for the same coordinates that of $\mathbf{y}$. Hence, we estimate the probability of a match at level j of the algorithm as follows:

$$\Pr\left[\left(\sum_{i=1}^{n}\mathbf{x}_i^{(j)}\right)_{l_1+\cdots+l_j} = \left(\sum_{i=1}^{n}\mathbf{y}_i\right)_{l_1+\cdots+l_j}\right] = \prod_{i=0}^{n}\Pr(X_n = -n+2i)^{\Pr(X_n=-n+2i)\cdot l_j}. \tag{8.2}$$

We acknowledge that this is a crude approximation of a complex probabilistic scenario, but our experimental results indicate that it yields surprisingly accurate predictions in practice.

For ease of representation, we write the success probability at each level j in the simplified form $(\Pr_j)^{l_j}$, where $\Pr_j$ is given by:

$$\Pr{}_j = \prod_{i=0}^{n}\Pr(X_n = 2i-n)^{\Pr(X_n=2i-n)} \quad \text{for} \quad n = 2^j. \tag{8.3}$$

As in the earlier discussion, in the final step we must consider all remaining coordinates of the target vector Y. Therefore, we define $l_q = m - \sum_{j=1}^{q-1} l_j$.

Once we have an estimate for the success probability of a collision, we can compute the expected list sizes at each level and thereby estimate the overall complexity of the algorithm.

8.2.3 Expected List Lengths

We know that at level j, the list $L_i^{(j)}$ contains $\mathbf{x}_i^{(j)}$ which have back-pointers to elements $\mathbf{x}_{2i-1}^{j-1}$ and $\mathbf{x}_{2i}^{(j-1)}$ of the two child lists used to form $L_i^{(j)}$ such that $\mathbf{x}_i^{(j)} = \mathbf{x}_{2i-1}^{j-1} + \mathbf{x}_{2i}^{j-1}$. This sum matches the corresponding partial sum of 2^j target vectors $\mathbf{y}$, restricted to the specific l_j coordinates. In general, for $i = 1, \ldots, k/2^j$ at each level $j = 1, \ldots, \log k$,

$$L_i^{(j)} = \left\{ \left(\mathbf{x}_i^{(j)}, \pi_i^{(j)} \right) \;\middle|\; \begin{array}{l} (\mathbf{x}_{2i-1}^{(j-1)}, -) \in L_{2i-1}^{(j-1)}, \quad (\mathbf{x}_{2i}^{(j-1)}, -) \in L_{2i}^{(j-1)}, \\ \mathbf{x}_i^{(j)} = \mathbf{x}_{2i-1}^{(j-1)} + \mathbf{x}_{2i}^{(j-1)}, \left(\mathbf{x}_i^{(j)} \right)_{l_1+\cdots+l_j} = \sum_{r=1}^{2^j} \left(\mathbf{y}_{(2^j)(i-1)+r} \right)_{l_1+\cdots+l_j} \\ \pi_i^{(j)} = \left(\pi_{2i-1}^{(j-1)} \parallel \pi_{2i}^{(j-1)} \right) \end{array} \right\}. \tag{8.4}$$

Even though we write it as $\left(\mathbf{x}_i^{(j)} \right)_{l_1+\cdots+l_j} = \sum_{r=1}^{2^j} \left(\mathbf{y}_{(2^j)(i-1)+r} \right)_{l_1+\cdots+l_j}$, as noted earlier in Remark 6.1, at each level j one only needs to check the condition for the last l_j entries. The analysis for the expected list length follows the same approach as in Section 6.3.2, but with updated success probabilities. We assume that all intermediate lists have the same expected length as the initial ones that allows us to give an asymptotic analysis.

The complexity of the algorithm can be analysed as follows. Using the probability estimation from Section 8.2.2, we have

$$(\Pr{}_1)^{l_1} = \Pr\left[(\mathbf{x}_1^{(0)} + \mathbf{x}_2^{(0)})_{l_1} = (\mathbf{y}_1 + \mathbf{y}_2)_{l_1} \right] = \left(\frac{1}{2} \right)^{3l_1/2}.$$

Set the expected list length at each level j to be m_j. Then, by linearity of expectation,

$$m_1 = \mathbb{E}\left[|L_1^{(1)}| \right] = |L_1^{(0)}||L_2^{(0)}| \cdot (\Pr{}_1)^{l_1} = m_0^2 \cdot \left(\frac{1}{2} \right)^{3l_1/2}.$$

Assuming $\mathbb{E}\left[|L_1^{(1)}| \right] = m_0$, we get $m_0 = 2^{3l_1/2}$.

At level 2, we consider the next l_2 coordinates of the vectors and filter pairs from $L_1^{(1)} \times L_2^{(1)}$ whose sum matches the partial sum $\mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3 + \mathbf{y}_4$ restricted to l_2 coordinates. Using a similar argument, we get

$$m_2 = \mathbb{E}\left[|L_1^{(2)}| \right] = |L_1^{(1)}||L_2^{(1)}| \cdot (\Pr{}_2)^{l_2} = m_0^2 \cdot (\Pr{}_2)^{l_2} \approx m_0.$$

Substituting $m_0 = 2^{3l_1/2}$, we solve for l_2:

$$l_2 = \frac{-3l_1}{2\log_2(\Pr{}_2)}.$$

In the final level, we require the expected length of the final list $L_1^{(3)}$ to be at least 1.

$$\mathbb{E}\left[|L_1^{(3)}| \right] = |L_1^{(2)}||L_2^{(2)}| \cdot (\Pr{}_3)^{l_3} = m_0^2 \cdot (\Pr{}_3)^{l_3} \geq 1.$$

Note that $l_3 = m - l_1 - l_2$, where m is the dimension of the vectors. Substituting the expressions for m_0 and l_2 into this relation allows us to solve for l_1 in terms of m. This determines the number of coordinates that should match at the first level. By recursively applying this process, we obtain the required initial list length m_0 to find a solution, as well as the number of coordinates l_j that should match at each level j.

General Setting. We extend the idea to any number of lists $k = 2^q$. This gives rise to a binary tree of depth $\log k = q$. At each internal node, we assume that the expected list length is equal to the initial list length m_0, and derive m_0 and the values of l_j in terms of l_1. More precisely, we obtain

$$m_0 = 2^{(\frac{3}{2})l_1} \quad \text{and} \quad l_j = \frac{-3l_1}{2\log_2(\mathrm{Pr}_j)} \quad \text{for } j = 2, \ldots, q-1,$$

where Pr_j is the approximate match probability defined in Equation (8.3).

At the final level q, we set $l_q = m - \sum_{j=1}^{q-1} l_j$, since the output vector must be of full dimension m. To ensure a solution to the k-sum problem is found with constant probability, we require the expected list length at level q to be at least one:

$$\mathbb{E}\left[|L_1^{(q)}|\right] = m_0^2 \,(\mathrm{Pr}_q)^{m-\sum_{j=1}^{q-1} l_j} \geq 1. \tag{8.5}$$

Substituting the expressions for m_0 and the l_j values into this equation and taking logarithms, we solve for l_1. The total sum of the l_j for $j = 1, \ldots, q-1$ is given by

$$\sum_{j=1}^{q-1} l_j = l_1 + \sum_{j=2}^{q-1} \frac{-3l_1}{2\log_2(\mathrm{Pr}_j)}.$$

Thus, the remaining dimension for level q becomes

$$m - \sum_{j=1}^{q-1} l_j = \frac{-3l_1}{\log_2(\mathrm{Pr}_q)},$$

which simplifies to the expression

$$l_1 = \frac{m}{1 - \frac{3}{2}\sum_{j=2}^{q-1} \frac{1}{\log_2(\mathrm{Pr}_j)} - \frac{3}{\log_2(\mathrm{Pr}_q)}}. \tag{8.6}$$

By substituting the computed value of l_1, we can determine m_0 and each l_j, thereby identifying the required initial list length and the number of coordinates that must match at each level $j = 1, \ldots, q$ for a final solution to exist with constant probability.

Algorithm 8 summarises how the k-list algorithm works to find the set of $\mathbf{x}_i^{(0)}$'s that add up to the target vector Y. We initialise with $\pi_i^{(0)} = \mathbf{x}_i^{(0)}$ and at each level j, $\pi_i^{(j)}$ acts as the back-pointer that keeps sufficient information to trace back to the original inputs at level 0.

Algorithm 8 k-List Algorithm for a given Y.

Input: $k = 2^q$, $L_i^{(0)}$ for $i = 1, \ldots, k$, $Y \in \mathbb{Z}_p^m$.
Output: $\mathbf{x}_1^{(0)}, \cdots, \mathbf{x}_k^{(0)}$, or $\perp$

for $j = 1, \ldots, q$ **do**
  Choose l_j such that $\sum_{j=1}^{q} l_j = m$.
  for $i = 1, \ldots, k/2^j$ **do**
    $L_i^{(j)} \leftarrow \emptyset$
    Sort $L_{2i-1}^{(j-1)}$ with respect to the first coordinate, $\mathbf{x}_{2i-1}^{(j-1)}$
    for each $(\mathbf{x}_{2i}^{(j-1)}, \pi_{2i}^{(j-1)}) \in L_{2i}^{(j-1)}$ **do**
      Find all $(\mathbf{x}_{2i-1}^{(j-1)}, \pi_{2i-1}^{(j-1)}) \in L_{2i-1}^{(j-1)}$ such that

$$(\mathbf{x}_{2i-1}^{(j-1)} + \mathbf{x}_{2i}^{(j-1)})_{l_1+\cdots+l_j} = \sum_{r=1}^{2^j} \left(\mathbf{y}_{(2^j)(i-1)+r}\right)_{l_1+\cdots+l_j}.$$

      for all such pairs **do**
        $\mathbf{x}_i^{(j)} \leftarrow \mathbf{x}_{2i-1}^{(j-1)} + \mathbf{x}_{2i}^{(j-1)}$
        $\pi_i^{(j)} \leftarrow \left(\pi_{2i-1}^{(j-1)} \parallel \pi_{2i}^{(j-1)}\right)$
        Add $(\mathbf{x}_i^{(j)}, \pi_i^{(j)})$ to $L_i^{(j)}$
      end for
    end for
  end for
end for
if $|L_1^{(q)}| \geq 1$ **then**
  return $\pi_1^{(q)}$
else
  return $\perp$
end if

8.3 Running time & Experimental Evidence

The running time and space complexity of this k-tree algorithm depends on the intermediate list sizes, as discussed in Chapter 6. For this analysis, we assume that all the lists have the same size m_0 throughout the internal steps.

This results in an algorithm for solving the new k-sum problem (Definition 8.1) with time and space complexity $\mathcal{O}(k \cdot m_0 \log m_0)$, where the list size is $\mathcal{O}(m_0)$, with $m_0 = 2^{3l_1/2}$ and l_1 expressed in terms of the dimension m and the matching probabilities, as given in Equation (8.6).

For the case $k = 8$, we observed that setting $\lceil l_1 \rceil = \lceil 0.343m \rceil$ allows us to find a solution to the 8-sum problem with non-trivial probability. With this choice, all lists have size approximately $m_0 \approx 2^{0.514m}$, and the algorithm runs in time and space complexity $\tilde{\mathcal{O}}(2^{0.514m})$ ignoring the constant and logarithmic factors.

We observe that the complexity improves as the number of lists increases ($k > 2^3$). In other words, if the target Y is a sum of more than 8 random vectors $\mathbf{y}_i \in \{-1, 1\}^m$, we can solve for Y using a list of random vectors $\mathbf{x}_i \in \{-1, 1\}^m$ in less than $2^{m/2}$ time and space.

In Table 8.1, we present the initial list lengths and the corresponding l_j values calculated asymptotically as a function of the vector dimension m for different numbers of lists $k = 2^q$. As shown, the initial list length m_0 decreases as the number of lists increases. Consequently, the new k-sum problem can be solved faster than square-root time for larger values of k.

k	l_1	l_2	l_3	l_4	l_5	l_6	l_7	$\log_2(m_0)$
4	$0.4037m$	$0.5963m$						$0.6055m$
8	$0.3427m$	$0.2532m$	$0.4041m$					$0.5141m$
16	$0.3018m$	$0.2230m$	$0.1780m$	$0.2972m$				$0.4528m$
32	$0.2727m$	$0.2015m$	$0.1608m$	$0.1343m$	$0.2307m$			$0.4091m$
64	$0.2510m$	$0.1854m$	$0.1480m$	$0.1236m$	$0.1061m$	$0.1860m$		$0.3764m$
128	$0.2340m$	$0.1728m$	$0.1379m$	$0.1152m$	$0.0989m$	$0.0867m$	$0.1544m$	$0.3510m$

Table 8.1: Examples for the initial list lengths and l_j values for $j = 1, \ldots, q$, for various numbers of lists k.

Recall from Section 8.1 that solving the $4-$sum version of the problem using the basic meet-in-the-middle algorithm requires lists of size $2^{0.5m}$, resulting in a total time complexity of $\mathcal{O}(2^m)$. In contrast, based on Table 8.1, the $4-$list algorithm we propose can solve the same problem in approximately $\mathcal{O}(2^{0.6m})$ time (ignoring the log factors), with larger list sizes of $2^{0.6m}$.

Therefore, we conclude that the k-list algorithm offers a substantial improvement over the classical meet-in-the-middle algorithm. For $k > 8$, the fact that the k-list algorithm

achieves time and space complexities below the square-root bound $\mathcal{O}\left(2^{m/2}\right)$ makes it superior to other known algorithms for high-dimensional problems where traditional approaches become infeasible.

8.3.1 Experimental Evidence

Heuristic 8.1. *The expected list lengths at the intermediate steps (except for the final list at level q) are equal to the initial list length m_0.*

In this section, we provide both theoretical and experimental evidence supporting Heuristic 8.1, which states that the expected list length m_j remains approximately equal to the initial list length m_0 across all levels of the algorithm. The experiments were conducted using small parameter sets to ensure computational feasibility.

We fixed small vector dimensions $m = 50, 60$ and considered the k-tree algorithm for $k = 2^7$ and $k = 2^9$. For these parameters, the rounded values of l_j and m_0 were calculated according to Table 8.1.

The theoretically expected list lengths were computed by the relation $m_j = m_0^2 \left(\Pr_j\right)^{l_j}$ and written in terms of m_0 at each level. As observed, the assumption $m_j = m_0$ does not hold exactly due to rounding errors introduced in the l_j values. Table 8.2 lists the specific l_j values used in the experiments alongside the corresponding theoretical estimates of m_j.

$m = 50$**,** $m_0 = 2^{18}, k = 2^7$

Rounded l_j	12	9	7	6	5	4	7
Expected m_j	$1.0m_0$	$0.8m_0$	$1.1m_0$	$0.8m_0$	$1.2m_0$	$3.5m_0$	18

$m = 50$**,** $m_0 = 2^{17}, k = 2^9$

Rounded l_j	11	8	6	5	5	4	4	3	4
Expected m_j	$1.4m_0$	$1.7m_0$	$3.3m_0$	$3.4m_0$	$0.6m_0$	$1.8m_0$	$0.4m_0$	$3.6m_0$	3594

$m = 60$**,** $m_0 = 2^{20}, k = 2^9$

Rounded l_j	13	10	8	6	5	5	4	4	5
Expected m_j	$1.4m_0$	$0.8m_0$	$0.8m_0$	$3.3m_0$	$4.8m_0$	$0.8m_0$	$3.5m_0$	$0.9m_0$	4920

Table 8.2: The rounded l_j and the Expected list lengths, m_j for $j = 1, \ldots, \log_2(k)$. This data is used in the experiment for Figure 8.2.

To compare the theoretical results with experimental findings, we test for merges in isolation for each level of the k-list algorithm for the same values of $k = 2^7, 2^9$ and vector dimensions $m = 50, 60$. For this experiment, we fixed the parameter l_j according to

Table 8.2, and considered vectors of length l_j, since this is the number of coordinates matched at a fixed level j.

The experiment proceeds as follows:

1. For a fixed j, generate two lists L_1 and L_2, each containing m_0 vectors of dimension l_j as follows.

$$L_i = \left\{ \sum_{j=1}^{2^{j-1}} x_j \;\mid\; x_j \leftarrow\$ \{-1, 1\}^{l_j} \right\} \quad \text{for } i = 1, 2$$

2. Sort L_1 lexicographically.

3. Repeat the following:

   (a) Generate the target vector Y such that $Y = \sum_{j=1}^{2^j} x_j$ where $x_j \leftarrow\$ \{-1, 1\}^{l_j}$.

   (b) Initialise $L^{(j)} = \{\}$

   (c) Check $\forall\ z_2 \in L_2$ is $z_1 = Y - z_2 \in L_1$ using a binary search.

   (d) If a match is found, append the pair (z_1, z_2) to $L^{(j)}$: $L^{(j)} = L^{(j)} \cup \{(z_1, z_2)\}$

   (e) Print $|L^{(j)}|$.

4. Compare the size of the list $|L^{(j)}|$ to the expected value $m_j = m_0^2 \, (\mathrm{Pr}_j)^{l_j}$ and categorise the results into three intervals:

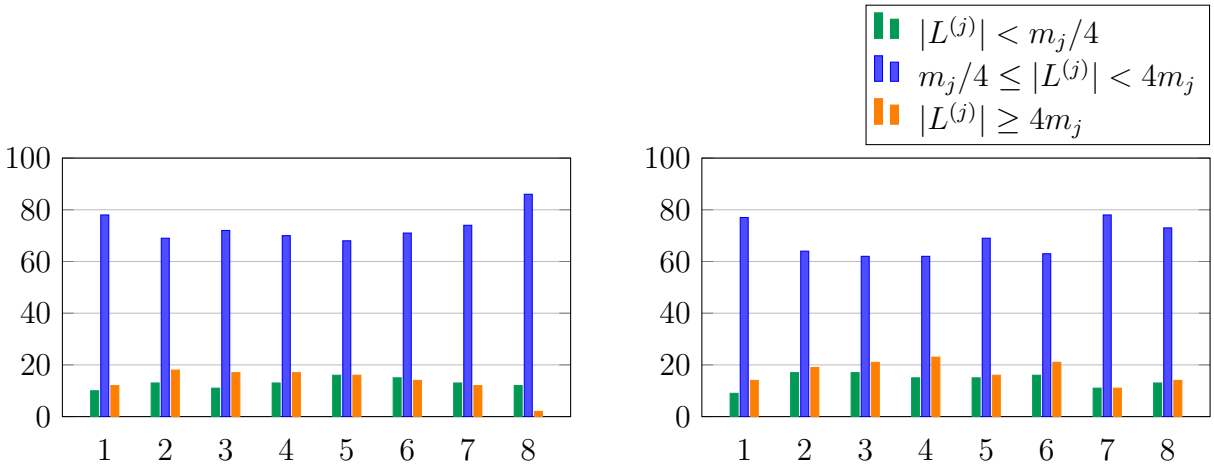
   - Within a factor of 4: $m_j/4 \leq |L^{(j)}| \leq 4m_j$,
   - Below expectation: $|L^{(j)}| < m_j/4$,
   - Above expectation: $|L^{(j)}| > 4m_j$.

Over 100 trials, we recorded the range of values attained by the experimental list lengths $|L^{(j)}|$ at different levels of the k-tree algorithm, in order to evaluate their deviation from the theoretical estimates m_j provided in Table 8.2. As shown in Figure 8.2, the experimental results demonstrate that, with high probability, the list lengths fall within the range $[m_j/4,\ 4m_j]$ at each level. In this figure, the x-axis corresponds to the levels of the k-tree algorithm, while the y-axis shows the frequency of list lengths falling into the respective intervals.

Across all nodes in the tree, some lists may be slightly larger than the expected value, while others may be slightly smaller. However, our experiments show that these deviations are not significant. In practice, the fluctuations tend to balance out when the lists are merged:

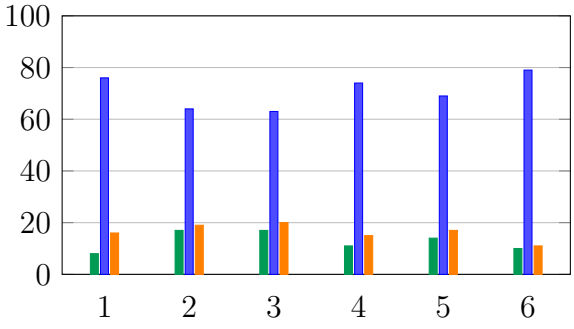
above-average-sized lists compensate for those that are below average, maintaining overall stability in the merging process. This experimental and theoretical comparison supports and validates our assumption that $m_j \approx m_0$ across levels.

It is also important to note that the expected list length m_j depends on the success probability calculations at each level. Therefore, the fact that the experimental outcomes closely match the theoretical expectations provides indirect validation of the accuracy of our average-case success probability estimates as well.



(a) $m = 50, k = 2^9, m_0 = 2^{17}$

(b) $m = 60, k = 2^9, m_0 = 2^{20}$



(c) $m = 50, k = 2^7, m_0 = 2^{18}$

Figure 8.2: Experimental results for $|L^{(j)}|$ for different m and k where $m_j = m_0^2(\mathrm{Pr}_j)^{l_j}$

8.4 Overall Average Running Time

All computations and experiments in Section 8.3 were carried out for a particular choice of fixed Y, as introduced in Section 8.2.2. These results illustrate that the running time of the algorithm depends inherently on the chosen target Y and may vary significantly between

instances. In extreme worst-case scenarios, for example, when $Y = (\pm k, \ldots, \pm k)$, the cost of finding a match can be exceptionally large; however, such extreme instances occur with negligible probability under the natural distribution of Y. At the opposite extreme, when Y is the zero vector, the problem essentially reduces to the zero-sum case discussed in Chapter 6. This naturally raises the question whether we can define and compute an *overall average running time* by averaging over many possible fixed Y's.

To formalise this idea, let $L = \{X_1, \ldots, X_N\}$ be a list of independent random vectors sampled from a distribution $\mathcal{D}$. Here $\mathcal{D}$ is the distribution of the sum of k independently and uniformly chosen vectors from $\{-1, 1\}^m$, that is,

$$X_i = \sum_{l=1}^{k} \mathbf{x}_{i,l}, \qquad \mathbf{x}_{i,l} \leftarrow\$ \{-1, 1\}^m.$$

Similarly, for each target we generate

$$Y_j = \sum_{l=1}^{k} \mathbf{y}_{j,l}, \qquad \mathbf{y}_{j,l} \leftarrow\$ \{-1, 1\}^m,$$

so that each Y_j is an independent sample from the same distribution $\mathcal{D}$. Assume throughout that $X_1, \ldots, X_N$ and $Y_1, \ldots, Y_M$ are independent and identically distributed (i.i.d.) samples from $\mathcal{D}$ and that the two collections are independent.

For a given target Y_1, the probability that a randomly chosen $X \in L$ equals Y_1 can be expressed as

$$\Pr(X = Y_1) = \frac{\text{total number of matches in } L}{N}.$$

Equivalently,

$$\text{total number of matches in } L = \sum_{i=1}^{N} \mathbf{1}_{\{X_i = Y_1\}},$$

where $\mathbf{1}_{\{X_i = Y_1\}} \in \{0, 1\}$ is the indicator of the event $X_i = Y_1$.

Repeating this experiment for M independently sampled targets $Y_1, \ldots, Y_M$, the *average matching probability* is

$$\frac{\text{total number of matches over all targets}}{NM} = \frac{1}{NM} \sum_{j=1}^{M} \sum_{i=1}^{N} \mathbf{1}_{\{X_i = Y_j\}}.$$

This can be rewritten as

$$\frac{1}{M} \sum_{j=1}^{M} \left(\frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{X_i = Y_j\}} \right).$$

For each fixed Y_j, the inner average $\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}_{\{X_i=Y_j\}}$ is the empirical probability that Y_j matches an element of the list L. By the law of large numbers, as N grows, this empirical probability converges to the probability $\Pr(X = Y_j)$ that a random $X \leftarrow\$\ \mathcal{D}$ equals a fixed target Y_j. Consequently,

$$\frac{1}{NM}\sum_{j=1}^{M}\sum_{i=1}^{N}\mathbf{1}_{\{X_i=Y_j\}} \;\approx\; \frac{1}{M}\sum_{j=1}^{M}\Pr(X = Y_j).$$

Applying the law of large numbers again, now for large M, this approximates to the probability that X equals a random $Y \leftarrow\$\ \mathcal{D}$.

$$\frac{1}{M}\sum_{j=1}^{M}\Pr(X = Y_j) \;\approx\; \Pr(X = Y),$$

where the last probability is precisely the *collision probability* between two independent random variables from the distribution $\mathcal{D}$. Thus, when both N and M are large, estimating the average matching probability over many fixed targets is asymptotically equal to computing the collision probability, i.e., the probability that two independent samples $X, Y \leftarrow\$\ \mathcal{D}$ are equal.

Since both X and Y are sums of k random $\{-1, 1\}^m$ vectors, the coordinates can take any value from $\{-k, -k+2, \ldots, k\}$. In a k-steps random walk, the probability that a single coordinate equals $p' \in \{-k, -k+2, \ldots, k\}$ is

$$\Pr(x = p') = 2^{-k}\binom{k}{\frac{k+p'}{2}}.$$

The standard collision probability calculations relies on the randomness and independence of both summands, and compute the collision probability as the sum of squared probabilities of all possible outcomes in a k-step random walk. Consequently, the probability that two vectors of dimension m sampled uniformly at random from $\mathcal{D}$ have a match is given by,

$$\Pr(X = Y) = \left(\sum_{p'}\Pr(x = p')^2\right)^m. \tag{8.7}$$

This compact formula provides a direct theoretical estimate for the average-case matching probability, and hence one can use it to compute the overall average running time of the algorithm. To verify the validity of our claim, we conduct the following experiment.

8.4.1 Experimental Evidence

This experiment estimates the average probability of collision between a randomly chosen target vector Y_j and a list of vectors X_i where both Y_j and each X_i are generated as the sum of k independent random vectors sampled uniformly from $\{-1, 1\}^m$. We compare the empirical collision probabilities with the theoretical estimates, and also evaluate both the actual and expected number of collisions.

Experiment Procedure:

1. Generate a list $L = \{X_1, X_2, \ldots, X_N\}$, where each vector X_i is constructed as $X_i = \sum_{l=1}^{k} x_{i,l}$ with $x_{i,l} \leftarrow\$ \{-1, 1\}^m$.

2. Fix a target vector (say Y_1) as $Y_1 = \sum_{l=1}^{k} y_{1,l}$ with $y_{1,l} \leftarrow\$ \{-1, 1\}^m$.

3. For a fixed target Y_1, count how many vectors in L exactly match Y_1. The collision probability for a fixed Y_1 is given by:

$$\Pr(X = Y_1) = \frac{\text{Total number of matches}}{N}.$$

4. **Experimental average:**
Repeat steps 1–3 for distinct target vectors $Y_j,\ j = 1, \ldots, M$, and compute the average probability of collision over all M targets:

$$\text{Avg. } \Pr(X = Y) = \frac{1}{M} \sum_{j=1}^{M} \Pr(X = Y_j). \tag{8.8}$$

The corresponding average number of matches per target Y_j is:

$$\text{Avg. number of matches} = \frac{\sum_{j=1}^{M} (\text{Total number of matcher per } Y_j)}{M}.$$

5. **Theoretical estimates:**
Assuming each X and Y are independent and randomly chosen samples from the same distribution, the theoretical collision probability $\Pr(X = Y)$ is calculated using Equation (8.7). The expected number of matches for a fixed Y is:

$$\mathbb{E}[\text{Theoretical Matches}] = N \cdot \Pr(X = Y).$$

6. Finally, compare the collision probabilities using theoretical estimates and empirical results given by Equation (8.7) and Equation (8.8) respectively. Moreover, the empirical average number of matches (from Step 4) were compared with the theoretical expected number of matches (from Step 5).

We perform this experiment for various parameter settings. The results, shown in Table 8.3, demonstrate that the empirical averages for both collision probability and number of matches closely align with the theoretical predictions. This agreement supports the conclusion that, for average running time analysis, standard collision probability estimates are reliable.

List Size (N)	Fixed Y_j (M)	Sum (k)	Dim. (m)	Avg Collision Pr (Experiment)	Collision Pr (Theory)	Avg Matches (Experiment)	Exp Matches (Theory)
300	300	10	5	0.000167	0.000170	0.050222	0.050947
300	300	2	4	0.019666	0.019775	5.899778	5.932617
500	300	8	8	0.000058	0.000057	0.029091	0.028679
500	300	16	5	0.000055	0.000054	0.027273	0.026843
300	500	6	6	0.000121	0.000132	0.036182	0.039536
300	500	16	5	0.000053	0.000054	0.015818	0.016106

Table 8.3: Comparison of empirical and theoretical collision probabilities and expected matches.

8.4.2 Average Running Time

We now use the collision probability to estimate the average running time of the algorithm. Consider again the basic meet-in-the-middle algorithm for $k = 4$. Let

$$X = \sum_{i=1}^{4} \mathbf{x}_i, \quad Y = \sum_{i=1}^{4} \mathbf{y}_i,$$

where X and Y are independent random variables drawn from the same distribution. Using the standard collision probability method, and from the properties of a random walk, the possible coordinate values for each vector are $\{-4, -2, 0, 2, 4\}$. In other words we have taken $k = 2^2$ steps in the random walk. Therefore, the collision probability for a single coordinate is

$$P_Y = \left(\frac{1}{2^4}\right)^2 + \left(\frac{4}{2^4}\right)^2 + \left(\frac{6}{2^4}\right)^2 + \left(\frac{4}{2^4}\right)^2 + \left(\frac{1}{2^4}\right)^2 = 0.27344.$$

For an m-dimensional vector, the probability of a coordinate-wise match is then $P_Y^m = (0.27344)^m$.

Following Section 8.1, assume we have two lists of size N. The probability that at least one match happens is

$$\Pr(\text{a match happens}) = 1 - (1 - P_Y^m)^{N^2}.$$

Using the approximation $(1-x)^a \approx e^{-ax}$ for small x, we obtain $\Pr(\text{a match happens}) \approx 1 - e^{-N^2 P_Y^m}$. To ensure a success probability of at least $1/2$, we require

$$e^{-N^2 P_Y^m} \leq \frac{1}{2} \quad \Rightarrow \quad N \geq \sqrt{\ln 2} \cdot P_Y^{-m/2}.$$

Since $P_Y = 0.27344$, this gives $N \approx (2^{0.93535m})$. The resulting total complexity of the algorithm is, therefore $\mathcal{O}(2^{0.93535m})$.

Compared to the fixed-target case from Section 8.1 where $Y = (\sqrt{k}, \ldots, \sqrt{k})$ led to a running time of $\mathcal{O}(2^m)$, the average-case running time is slightly better, but still significantly above square-root complexity.

Next, we show how these calculations can be used to determine the average running time of the k-tree algorithm over multiple target vectors Y_j. Recall that the k-tree algorithm proceeds through q levels, where $k = 2^q$. For a fixed target Y, the collision probability at each level j was computed in Section 8.2.2 using Equation (8.3). We can now replace this with the average collision probability, given by

$$\text{Average collision probability at level } j = \Pr{}_j(X = Y) = \sum_{p'} \Pr\big(x_n = p'\big)^2, \tag{8.9}$$

where $n = 2^j$ is the number of steps in the random walk and $p' \in \{-n, -n+2, \ldots, n\}$. Table 8.4 compares the two collision probabilities across the different levels j of the algorithm.

Once the collision probability at each level is determined, the subsequent calculations proceed exactly as in the fixed-Y case. An asymptotic analysis can be carried out by taking the list sizes to be equal at each stage and determining the matching coordinates l_j as described in Section 8.3. Table 8.5 illustrates how the change in collision probability affects the initial list length m_0, which determines the running time of the algorithm when it succeeds in finding a solution. The impact is more evident when the number of lists k is small than when it is large. Note that when $k = 4$, the k-tree algorithm achieves a better approximate running time of $\mathcal{O}(2^{0.56313m})$ compared to $\mathcal{O}(2^{0.93535m})$ for the basic

Level j	Collision Probability for a fixed Y Equation (8.3)	Avg. Collision Probability Equation (8.9)
1	0.35355	0.375
2	0.24475	0.27344
3	0.17144	0.19638
4	0.12103	0.13995
5	0.08556	0.09935
6	0.06049	0.07039
7	0.04278	0.04982

Table 8.4: Comparison of fixed and average collision probabilities for each level j.

meet-in-the-middle algorithm. The k-tree algorithm shows improved overall running time when moving from the fixed-target case to the average case.

While this provides insight into the algorithm's average performance when it always solves the instance, we emphasise that its actual running time always depends on the specific target vector Y. To account for this, we conducted experiments to determine the average running time for given vectors Y. For each choice of number of lists k and vector lengths m, we sampled 300 target vectors Y according to the distribution (sums of k vectors from $\{-1, 1\}^m$), calculated the corresponding collision probabilities and initial list sizes m_0. Then we obtained the average of m_0, which is asymptotically proportional to the average running time of the algorithm. Experiments were done for $m = 10, 20, 30, 50$ and $m = 100$, and in all the cases the experimental results closely match the theoretical analysis. The results for $m = 100$ are given in the third row of Table 8.5. It is evident that the these results are consistent with the theoretical estimates.

k	4	8	16	32	64	128
$\log_2(m_0)$ for a fixed Y	$0.6055m$	$0.5141m$	$0.4528m$	$0.4091m$	$0.3764m$	$0.3510m$
$\log_2(m_0)$ for average Y	$0.56313m$	$0.47780m$	$0.42157m$	$0.39340m$	$0.35674m$	$0.33287m$
$\log_2(m_0)$ experimental results	$0.57614m$	$0.48719m$	$0.42892m$	$0.38778m$	$0.35713m$	$0.33337m$

Table 8.5: Comparison of the initial list lengths

Remark 8.1. *Note that the collision probability computed via Equation (8.7) coincides with the success probability derived in Chapter 6 for the zero-sum version of the k-tree algorithm, where k is effectively replaced by $2k$. In other words, for average running time calculations,*

we can consider

$$\Pr\left(\sum_{i=1}^{k}\mathbf{x}_i = \sum_{i=1}^{k}\mathbf{y}_i\right) = \Pr\left(\sum_{i=1}^{2k}\mathbf{z}_i = \mathbf{0}\right),$$

where $\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i \xleftarrow{\$} \{-1, 1\}^m$ *are independent random vectors.*

8.5 Conclusion and Future Work

In this chapter, we introduced and analysed a new variant of the generalised birthday problem, motivated by its natural occurrence in the study of the rOM-ISIS problem. We formulated the problem as a novel k-sum problem: given a target vector $Y \in \mathbb{Z}^m$ formed by summing k random vectors from $\{-1, 1\}^m$, the goal is to reconstruct Y as the sum of k other independently chosen vectors from the same distribution. This restriction of having a fixed target differs from the classical setting of Wagner's algorithm, introducing unique algorithmic and analytical challenges.

We observed that the problem remains significantly hard even for the simplest case of a 4-list version using a meet-in-the-middle strategy. To address this, we adapted Wagner's k-list algorithm to our setting. We described a tree-based algorithm and studied its expected behaviour both theoretically and experimentally. Our analysis was done first for a fixed Y considering an example and later we obtained an overall average running time. In both instances, we observed that the new k-tree algorithm improved the complexity substantially compared to naive approaches.

To the best of our knowledge, the problem introduced in this chapter has not been studied before. Through this thesis we made considerable progress in understanding the distribution of a given target, and its effect on analysing the success probability in finding a solution to the new k-sum problem. We also make positive progress towards understanding the overall running time over many possible targets. However, we believe that several directions remain open for further exploration.

Future Work

We outline a few immediate directions for future research:

- **Alternate Success Probability Analysis.** Investigating alternative methods for estimating success probabilities that do not rely on decomposing the target vector Y could provide new insights, and analysing their impact on the algorithm's running time would be valuable.

- **Understanding the Distribution of Y.** Since Y is a sum of k i.i.d. ± 1 vectors, we studied the distribution as a Rademacher distribution and use the properties of a symmetrical random walk in the analysis. A deeper study of the distribution of the target vector Y (for example, as a multivariate normal distribution) could yield better understanding of the problem's inherent difficulty. A more precise characterisation of this distribution may also help assess its effect on the algorithm's complexity.
- **Implementation and Empirical Validation.** Implementing the proposed algorithm and empirically validating the theoretical estimates could confirm assumptions made during analysis, reveal practical bottlenecks, and guide refinements to both the algorithm and its complexity estimates.

We believe this study opens a new and interesting research direction, and we hope it will motivate further investigation into the combinatorial and algorithmic aspects of this problem, as well as its potential applications in cryptography.

Chapter 9

Conclusion and Future Work

This thesis addressed two main research problems: the design and analysis of a lattice-code scheme, and the study of novel combinatorial problems arising from a new hardness assumption. Together, these investigations advance the understanding of previously underexplored areas in post-quantum cryptography.

9.1 Concluding Remarks

In the first part of the thesis we investigated the LLXY scheme, which is a McEliece cryptosystem constructed using a code coming from lattices, known as relation lattices. We studied the lattice family underlying the LLXY scheme and showed that its original encryption construction is not secure against adaptive Chosen Ciphertext Attacks (CCA). We proposed a reformulation of the scheme as a Niederreiter-style Key Encapsulation Mechanism (KEM), which enables more compact ciphertexts and straightforward CCA security using standard transformations. Our security evaluation incorporated the state-of-the-art developments in both lattice-based and decoding attacks, providing guidance on safe parameter choices and highlighting the scheme's flexibility with respect to error distributions. Beyond these proposals, we introduced a new formulation of the relation lattice, offering additional structural insight that could benefit future cryptanalysis.

The second part of the thesis focused on algorithmic problems related to the generalised birthday problem. We first introduced a new variant of this problem over $\{-1, 1\}^m$ and showed that it is harder than the classical generalised birthday problem defined over $\{0, 1\}$ vectors. We then proposed Wagner-style k-tree algorithms that outperform naive meet-in-the-middle approaches, together with theoretical analyses and complexity estimates. Building on this framework, we applied these algorithmic

techniques to the recently proposed randomised one-more Inhomogeneous Short Integer Solution (rOM-ISIS) assumption, demonstrating that additional valid solutions can be generated efficiently, albeit with complexity exponential in the dimension. Although this does not currently compromise the associated blind signature scheme under the published parameter sets, it suggests that the underlying assumption may be weaker than originally intended, as it is susceptible to new types of attacks that do not apply to the standard one-more ISIS problem. Finally, we introduced and analysed a generalised variant of the k-sum problem with a fixed target vector. To the best of our knowledge, this constitutes the first systematic study of the problem, including heuristic analyses as well as directions for empirical validation.

Overall, by enhancing both the design of lattice–code schemes and the analysis of novel algorithmic problems, we believe we have contributed in strengthening the post-quantum cryptographic toolkit and hope that the methods, insights, and open questions presented here will inspire further research in these areas.

9.2 Future Work

Several promising research directions emerge from our study are listed below.

Lattice–Code Based Cryptography

- A more systematic investigation of alternative error distributions in the LLXY KEM could yield improved trade-off between efficiency and security.

- The underlying polynomial lattice structure warrants deeper cryptanalytic study, particularly to determine whether specialised structural attacks can be mounted against the scheme.

- An actual implementation of the LLXY KEM would enable evaluation of its practical viability and provide a comparative benchmark against NIST standardisation candidates.

- The LLXY scheme demonstrates that combining lattice-based and code-based approaches can give rise to secure hybrid constructions taking advantages from both directions, and it is worth to further research on such cross-directional work.

Algorithmic Problems and Combinatorial Cryptanalysis

- Further study of the k-sum problem with $\{-1, 1\}$ vectors, particularly in the fixed-target setting, could lead to tighter complexity analyses and the design of more efficient algorithms.
- Implementing and empirically validating the proposed k-tree algorithms would provide practical insights, confirm theoretical predictions, and guide refinements beyond heuristic analysis.
- Exploring further cryptographic contexts where these newly introduced algorithms can be applied may open new avenues in combinatorial cryptanalysis.

rOM-ISIS Assumption and Blind Signatures

- The rOM-ISIS assumption warrants further scrutiny: the cryptanalysis presented in the original paper is very limited, and a more rigorous analysis would provide a clearer understanding of its security and its suitability for use in cryptographic schemes.
- The blind signature scheme currently relies on highly conservative parameters that are impractical. Implementing the scheme and carrying out a detailed security analysis would enable the selection of more practical parameter sets.
- Investigating a combinatorial attack based on a single-list approach for finding preimage vectors whose sum is zero, as an alternative to the k-list method studied here. Carrying out such a detailed analysis will clarify whether this approach can reduce the number of required preimage queries while maintaining comparable complexity.

Bibliography

[ABB+22] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Santosh Ghosh, Shay Gueron, Tim Güneysu, et al. BIKE: Bit Flipping Key Encapsulation. *NIST PQC Round*, 2022.

[AD21] Martin Albrecht and Léo Ducas. Lattice attacks on NTRU and LWE: a history of refinements. *Cryptology ePrint Archive*, 2021.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key Exchange-A new hope. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 327–343, 2016.

[AG25] Pabasara Athukorala and Steven D Galbraith. Breaking and Improving a Lattice-Code-based cryptosystem by Li, Ling, Xing and Yeo. *IEEE Transactions on Information Theory*, 71(9):6857–6869, 2025.

[AGVW17] Martin R Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In *Asiacrypt 2017*, pages 297–322. Springer, 2017.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[AKSY22] Shweta Agrawal, Elena Kirshanova, Damien Stehlé, and Anshu Yadav. Practical, round-optimal lattice-based blind signatures. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 39–53, 2022.

[Alb17] Martin R Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In *Eurocrypt 2017*, pages 103–129. Springer, 2017.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[Bab86] László Babai. On Lovász'lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[BBB+17] Magali Bardet, Elise Barelli, Olivier Blazy, Rodolfo Canto Torres, Alain Couvreur, Philippe Gaborit, Ayoub Otmani, Nicolas Sendrier, and Jean-Pierre Tillich. Big quake binary Goppa quasi-cyclic key encapsulation. hal-01671866v2, 2017.

[BBD09] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors. *Post-Quantum Cryptography*, volume 1. Springer, 2009.

[BBF+19] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila. Hybrid key encapsulation mechanisms and authenticated key exchange. In *PQCrypto 2019*, pages 206–226. Springer, 2019.

[BCGY24] Foteini Baldimtsi, Jiaqi Cheng, Rishab Goyal, and Aayush Yadav. Non-interactive Blind Signatures: Post-quantum and Stronger security. In *Asiacrypt 2024*. Springer, 2024.

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *Eurocrypt 2011*, pages 364–385. Springer, 2011.

[BCL+17] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic McEliece: conservative code-based cryptography. *NIST submissions*, 1(1):1–25, 2017.

[BDK+18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European symposium on security and privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

[BF11] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *PKC 2011*, pages 1–16. Springer, 2011.

[BGLS19] Shi Bai, Steven D Galbraith, Liangze Li, and Daniel Sheffield. Improved combinatorial algorithms for the Inhomogeneous Short Integer Solution Problem. *Journal of Cryptology*, 32:35–83, 2019.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BHH+19] Nina Bindel, Mike Hamburg, Kathrin Hövelmanns, Andreas Hülsing, and Edoardo Persichetti. Tighter proofs of CCA security in the quantum random oracle model. In *TCC 2019*, volume 11892, pages 61–90. Springer, 2019.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *Eurocrypt 2012*, pages 520–536. Springer, 2012.

[BL05] Thierry P Berger and Pierre Loidreau. How to mask the structure of codes for a cryptographic use. *Designs, Codes and Cryptography*, 35:63–79, 2005.

[BLNS23] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Alessandro Sorniotti. A framework for practical anonymous credentials from lattices. In *Crypto 2023*, pages 384–417. Springer, 2023.

[BLP08] Daniel J Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In *PQCrypto*, pages 31–46, 2008.

[BM17a] Leif Both and Alexander May. The approximate k-list problem. *IACR Transactions on Symmetric Cryptology*, pages 380–397, 2017.

[BM17b] Leif Both and Alexander May. Optimizing BJMM with nearest neighbors: full decoding in $2^{2n/21}$ and McEliece security. In *WCC workshop on coding and cryptography*, volume 214, 2017.

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In *PQCrypto 2018*, pages 25–46. Springer, 2018.

[BMVT78] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information theory*, 24(3):384–386, 1978.

[BP18] Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. *Cryptology ePrint Archive*, page 526, 2018.

[BSW16] Shi Bai, Damien Stehlé, and Wen Weiqiang. Improved reduction from the Bounded Distance Decoding Problem to the Unique Shortest Vector Problem in Lattices. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2016.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto 82*, pages 199–203. Springer, 1983.

[Che13] Yuanmi Chen. *Réduction de réseau et sécurité concrete du chiffrement completement homomorphe*. PhD thesis, Paris 7, 2013.

[CHHS19] Jung Hee Cheon, Minki Hhan, Seungwan Hong, and Yongha Son. A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret LWE. *IEEE Access*, 7:89497–89506, 2019.

[CJ04] Jean-Sebastien Coron and Antoine Joux. Cryptanalysis of a provably secure cryptographic hash function. *Cryptology ePrint Archive*, 2004.

[CN11] Yuanmi Chen and Phong Q Nguyen. BKZ 2.0: Better lattice security estimates. In *Asiacrypt 2011*, pages 1–20. Springer, 2011.

[CP91] Paul Camion and Jacques Patarin. The Knapsack Hash function proposed at Crypto'89 can be broken. In *Eurocrypt 1991*, pages 39–53. Springer, 1991.

[CR84] Benny Chor and Ronald L. Rivest. A Knapsack type Public key Cryptosystem based on Arithmetic in Finite Fields. In *Crypto 1984*, volume 196, pages 54–65. Springer, 1984.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive Chosen Ciphertext Attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[CWC+24] Jinrong Chen, Yi Wang, Rongmao Chen, Xinyi Huang, and Wei Peng. Tighter proofs for PKE-to-KEM transformation in the quantum random oracle model. In *Asiacrypt 2024*, pages 101–133. Springer, 2024.

[DEL25] Léo Ducas, Lynn Engelberts, and Johanna Loyer. Wagner's Algorithm provably runs in subexponential time for SIS^{∞}. In *Crypto 2025*, pages 353–384. Springer, 2025.

[Den03] Alexander W Dent. A designer's guide to KEMs. In *IMA International Conference on Cryptography and Coding*, pages 133–151. Springer, 2003.

[Din19] Itai Dinur. An algorithmic framework for the Generalized Birthday Problem. *Designs, Codes and Cryptography*, 87:1897–1926, 2019.

[DP19] Léo Ducas and Cécile Pierrot. Polynomial time Bounded Distance Decoding near Minkowski's bound in discrete logarithm lattices. *Designs, Codes and Cryptography*, 87:1737–1748, 2019.

[DP23] Léo Ducas and Ludo N Pulles. Does the dual-sieve attack on learning with errors even work? In *Crypto, 2023*, pages 37–69. Springer, 2023.

[Duj04] Andrej Dujella. Continued fractions and RSA with small secret exponents. *Tatra Mountains Mathematical Publications*, 29(101):101–112, 2004.

[Dum91] Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52. Moscow, 1991.

[Ess23] Andre Esser. Revisiting nearest-neighbor-based Information Set Decoding. In *IMA International Conference on Cryptography and Coding*, pages 34–54. Springer, 2023.

[FHK+18] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang, et al. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. *Submission to the NIST's post-quantum cryptography standardization process*, 36(5):1–75, 2018.

[Gal12] Steven D Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[GN08a] Nicolas Gama and Phong Q Nguyen. Finding short lattice vectors within Mordell's inequality. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 207–216, 2008.

[GN08b] Nicolas Gama and Phong Q Nguyen. Predicting lattice reduction. In *Eurocrypt 2008*, pages 31–51. Springer, 2008.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206, 2008.

[GvVW17] Florian Göpfert, Christine van Vredendaal, and Thomas Wunderer. A hybrid lattice basis reduction and quantum search attack on LWE. In *PQCrypto 2017*, pages 184–202. Springer, 2017.

[Han23] Lucjan Hanzlik. Non-interactive blind signatures for random messages. In *Eurocrypt 2023*, pages 722–752. Springer, 2023.

[HG07] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *Crypto 2007*, pages 150–169. Springer, 2007.

[HGJ10] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Eurocrypt 2010*, pages 235–256. Springer, 2010.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *TCC 2017*, pages 341–371. Springer, 2017.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number theory symposium*, pages 267–288. Springer, 1998.

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.

[JKL24] Antoine Joux, Hunter Kippen, and Julian Loss. A concrete analysis of Wagner's k-list algorithm over $\mathbb{Z}_p$. *Cryptology ePrint Archive*, 2024.

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures. In *Crypto 1997*, pages 150–164. Springer, 1997.

[JZC+18] Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. In *Crypto 2018*, pages 96–125. Springer, 2018.

[JZM21] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. On the non-tightness of measurement-based reductions for key encapsulation mechanism in the quantum random oracle model. In *Asiacrypt 2021*, pages 487–517. Springer, 2021.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.

[Kir11] Paul Kirchner. Improved Generalized Birthday attack. *Cryptology ePrint Archive*, 2011.

[KKN+26] Alexander Karenin, Elena Kirshanova, Julian Nowakowski, Eamonn W Postlethwaite, Ludo N Pulles, Fernando Virdia, and Paul Vié. Cool+ cruel= dual, and new benchmarks for sparse LWE. In *Eurocrypt, 2025*, pages 304–333. Springer, 2026.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

[KM06] Sara Khodadad and Michael Monagan. Fast rational function reconstruction. In *Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 184–190, 2006.

[Lap21] Oleksandra Lapiha. Comparing lattice families for Bounded Distance Decoding near Minkowski's bound. *Cryptology ePrint Archive*, 2021.

[LDW94] Yuan Xing Li, Robert H Deng, and Xin Mei Wang. On the equivalence of Mceliece's and Niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1):271–273, 1994.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In *International conference on security and cryptography for networks*, pages 348–359. Springer, 2006.

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261(4):515–534, 1982.

[LLXY17] Zhe Li, San Ling, Chaoping Xing, and Sze Ling Yeo. On the Closest Vector Problem for lattices constructed from polynomials and their cryptographic applications. *arXiv preprint arXiv:1710.02265*, 2017.

[LLXY20] Zhe Li, San Ling, Chaoping Xing, and Sze Ling Yeo. On the Bounded Distance Decoding problem for lattices constructed and their cryptographic applications. *IEEE Transactions on Information Theory*, 66(4):2588–2598, 2020.

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem. *Crypto 2009*, page 577, 2009.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Eurocrypt 2010*, pages 1–23. Springer, 2010.

[LWXZ14] Mingjie Liu, Xiaoyun Wang, Guangwu Xu, and Xuexin Zheng. A note on BDD problems with λ_2-gap. *Information Processing Letters*, 114(1-2):9–12, 2014.

[LXY19] Zhe Li, Chaoping Xing, and Sze Ling Yeo. Reducing the key size of McEliece cryptosystem from automorphism-induced Goppa codes via permutations. In *PKC-2019*, pages 599–617. Springer, 2019.

[Lyu05a] Vadim Lyubashevsky. On random high density subset sums. *Electronic Colloquium on Computational Complexity (ECCC)*, 12(007), 2005.

[Lyu05b] Vadim Lyubashevsky. The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 378–389. Springer, 2005.

[MAB+18] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loıc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and IC Bourges. Hamming quasi-cyclic (HQC). *NIST PQC Round*, 2(4):13, 2018.

[Man02] Yannis Manolopoulos. Binomial coefficient computation: recursion or iteration? *ACM SIGCSE Bulletin*, 34(4):65–67, 2002.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic coding theory. *Coding Thv*, 4244:114–116, 1978.

[MG02] Daniele Micciancio and Shafi Goldwasser. *Complexity of lattice problems: a cryptographic perspective*, volume 671. Springer Science & Business Media, 2002.

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *computational complexity*, 16:365–411, 2007.

[MO15] Alexander May and Ilya Ozerov. On computing Nearest Neighbors with applications to decoding of binary linear codes. In *Eurocrypt 2015*, pages 203–228. Springer, 2015.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Eurocrypt 2012*, pages 700–718. Springer, 2012.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM journal on computing*, 37(1):267–302, 2007.

[MS12] Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. *Journal of cryptology*, 25:349–382, 2012.

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):157–166, 1986.

[NS15] Ivica Nikolić and Yu Sasaki. Refinements of the k-tree algorithm for the Generalized Birthday Problem. In *Asiacrypt 2015*, pages 683–703. Springer, 2015.

[OS09] Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In *Post-quantum cryptography*, pages 95–145. Springer, 2009.

[Pei16] Chris Peikert. A Decade of Lattice Cryptography. *Foundations and trends® in theoretical computer science*, 10(4):283–424, 2016.

[Per12] Edoardo Persichetti. *Improving the efficiency of Code-based Cryptography*. PhD thesis, University of Auckland, 2012.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[Sch03] Claus Peter Schnorr. Lattice reduction by random sampling and birthday methods. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 145–156. Springer, 2003.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1):181–199, 1994.

[Sel95] B. I. Selivanov. On waiting time in the scheme of random allocation of coloured particles. *Discrete Mathematics and Applications*, 5(1):73–82, 1995.

[Sha08] Andrew Shallue. An improved multi-set algorithm for the dense subset sum problem. In *International Algorithmic Number Theory Symposium*, pages 416–429. Springer, 2008.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. IEEE, 1994.

[SP97] Rainer Storn and Kenneth Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359, 1997.

[SS81] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2}), S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM journal on Computing*, 10(3):456–464, 1981.

[SS92] Vladimir Michilovich Sidelnikov and Sergey O Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 2(4):439–444, 1992.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In *International colloquium on coding theory and applications*, pages 106–113. Springer, 1988.

[Str10] Falko Strenzke. How to implement the public key operations in code-based cryptography on memory-constrained devices. *Cryptology ePrint Archive*, 2010.

[TSG25] Lili Tang, Yao Sun, and Xiaorui Gong. Revisiting the Generalized Birthday Problem and Equihash: Single or K Lists? *Cryptology ePrint Archive*, 2025.

[Wag02] David Wagner. A Generalized birthday problem. In *Crypto 2002*, pages 288–304. Springer, 2002.

[Wal18] Michel Waldschmidt. Diophantine approximations and continued fractions. In *Proceedings of the Roman Number Theory Association*, volume 9, 2018.

[Wun18] Thomas Wunderer. *On the security of Lattice-Based Cryptography against Lattice Reduction and Hybrid attacks*. PhD thesis, Technische Universität Darmstadt, 2018.